Improving History-Based Change Recommendation Systems for Software Evolution

Ph.D. Dissertation

Thomas Gramstad Rolfsnes

Dissertation submitted July, 2017

© Thomas Gramstad Rolfsnes, 2017

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 1885

ISSN 1501-7710

All rights reserved. No part of this publication may be
reproduced or transmitted, in any form or by any means, without permission.

Cover: Hanne Baadsgaard Utigard.
Print production: Reprosentralen, University of Oslo.

Abstract

The software that we depend on every day is constantly changed. These changes are necessary to comply with shifting user requirements, keeping a competitive advantage, adapting to changes in other software, and to fix the ever-present bugs. It is crucial that the impact of these changes is well understood, as failure to do so may well lead to additional bugs being introduced, which may directly affect the longevity and success of the software.

In order to understand the impact of a change, the parts of the software which is affected by the change must be uncovered. This thesis explores through five papers how the *change history* of software can be leveraged to address this challenge. From a change history, it is possible to identify the files and methods that typically *change together*. Furthermore, these *change patterns* can successfully be used to derive *association rules* which can predict the impact of future changes.

As with all recommendation systems, a danger is always that parts of a recommendation are wrong or missing (false positives and negatives). This thesis presents steps towards reducing such issues for change recommendation systems utilizing change patterns and association rule mining. The contributions with respect to each paper are as follows: **(A)** A targeted association rule mining algorithm for filtering out irrelevant association rules **(B)** An approach for aggregating association rules, increasing overall recommendation precision **(C)** A study of parameters for association rule mining in this context **(D)** A study of the effects of change history length and age on recommendation precision **(E)** And lastly, an approach for predicting whether a change recommendation is relevant. These contributions help drive forward a field which has long been dominated by approaches based on static and dynamic analysis. By building on top of version control systems, our approach has considerably less overhead and is inherently language agnostic.

Acknowledgements

4 years ago, I never thought I would do a PhD. Now, I'm sitting in front of 200 pages that I'm proud of. This has only been possible because of a group of amazing people, and I am sincerely grateful for all the time you have given me. I must express my deepest gratitude to my main supervisor, Leon Moonen, you have helped me develop myself, both as a researcher and a person. To Razieh Behjati, which has been with me since my master thesis, your feedback along the way has been invaluable. To Stefano Di Alesio, I'll probably have dreams about our many whiteboard sessions. And to Dave Binkley, you were the missing member in our team of Avengers, we couldn't have done it without you.

And to my wife, Ingrid, one should think that finishing a degree in clinical psychology, and giving birth to our first son with the second on the way, would be more than enough for a single person to handle. But through all of this you have been my greatest cheerleader, and have always been there when I needed you.

To my son, Elias, you are my greatest motivation.

Contents

Part I

Summary

Summary

1 Introduction

Every instant, somewhere in the world, a line of code is changed. Bugs are fixed and new features implemented. Meanwhile, code is also inherently interconnected with pieces of logic requiring other pieces of logic in complex networks. Thus every time a line of code is changed we incur the risk of invalidating the intended logic encapsulated by the program. At the same time, non code artifacts such as *configuration files* or *documentation* are also in need of synchronization. Complicating the matter even further, software of significant size is seldom worked on by a single developer, and each developer may have a different *mental model* of the software system. Furthermore, as the software grows larger it becomes progressively harder for any single developer to retain complete knowledge of the intricacies of interacting artifacts.

With the advent of electronic computers, developers early on saw the need for *version control* to track changes to their software over time. Case in point, the first system for version control, SCCS, stems all the way back to 1972 [1]. As their most basic feature, version control systems makes it possible to retrieve any previous version of a particular file. In modern times however, version control systems are a critical component in enabling multiple developers to seamlessly work on the same code base. Part of this stems from the central concept of a *change set* which describes exactly the lines that were added, changed or deleted across multiple files. Typically, each change set is also accompanied with a written message summarizing the intention behind the changes. Thus version control systems effectively supports *best practices* by helping developers think about code modifications as *isolated units of work*.

As software evolves, more and more change sets are indexed by the version control system to slowly build up a *change history*. Within the change history, we can trace the evolution of every artifact, and of particular interest in our case, we can identify patterns describing how artifacts typically change together. For example, we might observe that whenever $methodA()$ is changed, then methods $methodB()$ and $methodC()$ are also typically changed.

Through these patterns we can communicate valuable information to the software developer. We can for example calculate the potential impact that a set of changes has, or inform the developer about potential "next steps". Such recommendations can help maintain the mental model for an experienced developer, or help build the model from scratch for a developer new to the system. Consider the following simple example:

Example 1 (Identifying Change Patterns). *A video chat system has been built in such a way that certain changes to the* close_connection() *method requires that the* open_connection() *method also needs to be updated. Furthermore, the configuration file* connection_config *on occasion also needs to be updated. Given these couplings, the change history might look something like this:*

ID	*Change set*
⋮	⋮
100	`close_connection(),open_connection()`
⋮	⋮
104	`close_connection(),open_connection(),connection_config`
105	`close_connection()`
⋮	⋮

The line with ID = 100, describes that close_connection() *and* open_connection() *were changed together in a change set. A little later, the same changes are repeated, but now the* connection_config *file is also changed. Then, in the next change set,* close_connection() *is changed by itself. From only these few change sets many patterns can be identified, for example:*

- *When* connection_config *is changed, both* close_connection() *and* open_connection() *is also changed.*
- *When* open_connection() *is changed,* close_connection() *is changed.*
- *When* close_connection() *is changed,* open_connection() *and* connection_config *is sometimes changed.*

Example 1 highlights how patterns can be mined from change histories, such patterns are called *evolutionary couplings* as they emerge from how artifacts change together over time. On the one hand, evolutionary couplings provide access to the inherent knowledge that developers have of a software system. On the other hand, these couplings may also on occasion be erroneous as a result of coming from a human source. For example, a developer might forget to make a required modification resulting in the corresponding change set missing a related artifact. Alternatively, a developer might erroneously add

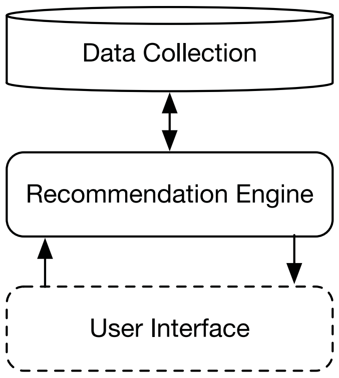


Fig. 1: The architecture of a recommender system in its most basic form

artifacts relating to *different* implementation tasks to the same change set, thus erroneous couplings might be established. Throughout this thesis we will discuss several techniques for reducing the impact of erroneous couplings, as well as quantifying their inherent uncertainty. These contributions can be seen as part of a greater effort which seeks to improve *Recommender Systems for Software Engineering* (RSSE). The intent behind RSSE has succintly been defined as follows:

> *An RSSE is a software application that provides information items estimated to be valuable for a software engineering task in a given context.* [2]

Existing RSSE aids in tasks such as recommending expert consultants [3] and developers [4], to more code related recommendations such as identifying buggy code [5] or code related to a change [6]. Robillard et al. found that the architecture of an RSSE must consists of at least 3 components, as visualized in Figure 1: **(1)** A data collection mechanism which provides the data underlying the recommendations. **(2)** The recommendation engine it self which processes input against the collected data and generates recommendations. **(3)** A user interface which facilitates input for the recommendation engine and displays the resulting recommendations. The research effort presented in this thesis instantiates the architecture of Figure 1 in the following manner:

Example Tasks: Quantifying the impact of a change, or recommending relevant parts of a code-base based on a change.

Data Collection Mechanism: The underlying data is mined from software repositories, or more specifically, version control software such as `git`. The final form of the data is a sequence of change sets capturing the change history of a software system. In paper E we extend the data collection to also encompass previous recommendations with their context.

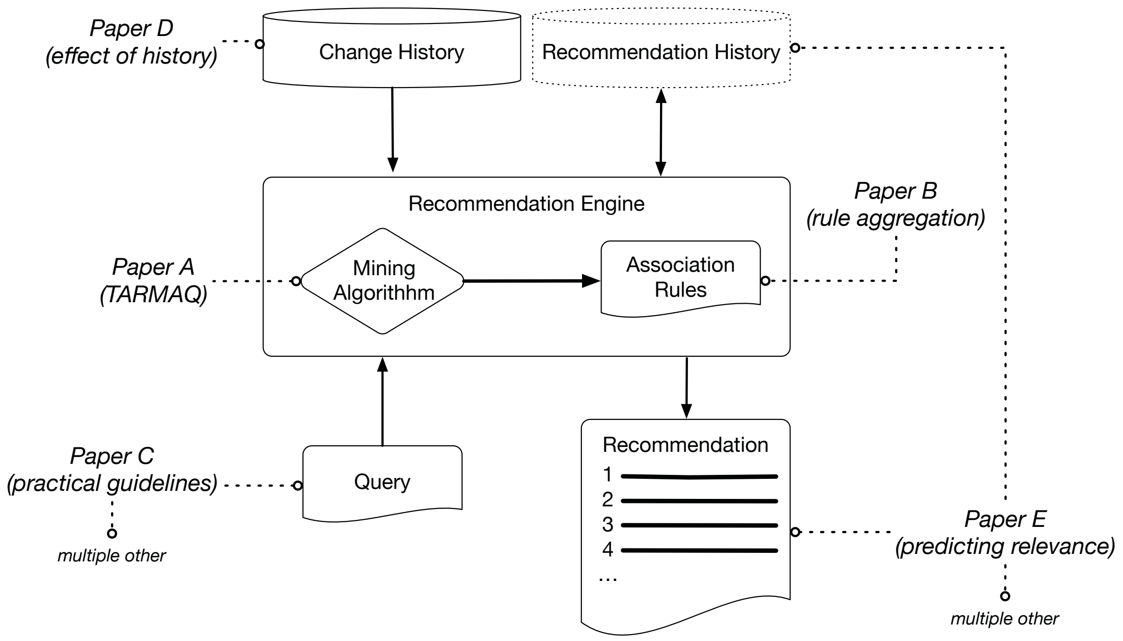


Fig. 2: Each paper addresses part(s) of an RSSE

Recommendation Engine: The engine processes an input query against the obtained change history to identify interesting patterns. The pattern search is based on association rule mining.

User Interface: The recommendation engine accepts queries in the form of a set of artifacts, and outputs a ranked list of relevant artifacts for that query.

Each paper either seeks to improve the quality of the output recommendation in some manner, or explores in which scenarios we should expect a recommendation to be accurate. Figure 2 presents the RSSE explored in this thesis, using dashed lines to designate the part of the system each paper is concerned with.

The remainder of this summary is organized as follows: The next section briefly introduces association rule mining. Then in section 3-7 the problem addressed by each paper is discussed along with findings. Finally in section 8 we discuss the overarching issue of *change granularity*, and describe our experiment architecture in section 9.

2 Primer on Association Rule Mining

Agrawal et al. introduced the concept of *association rule mining* as the discipline aimed at inferring relations between *artifacts* of a data set [7]. *Association rules* are implications of the form $A \rightarrow B$, where A is referred to as the *antecedent*, B as the *consequent*, and A and B are disjoint sets. For example, consider the classic application of analyzing shopping cart data; if multiple transactions include bread and butter then a potential association rule is *bread* $\rightarrow$ *butter*. This rule can be read as *"if you buy bread, then you are also likely to buy butter."*

In the context of mining evolutionary coupling from historical change sets, *artifacts* are not shopping items, but rather represents changes to *files*, *methods*, *classes* and so on. A set of artifacts (e.g., a *commit*), is called a *transaction*, but is also referred to as a *change set* on occasion. The sequence of transactions that naturally emerges as software evolves is called a *change history*. Given a change history, we can then *mine* association rules such as the ones in Example 1, e.g., $close\_connection() \rightarrow open\_connection()$.

An association rule is however of less use if we cannot quantify its importance somehow, for this purpose we have *interestingness measures*. For example, the rule $close\_connection() \rightarrow open\_connection()$ from Example 1 has a *frequency* of 2, since $close\_connection()$ and $open\_connection()$ are changed together in two transactions. Over 40 other interestingness measures have been introduced [8, 9].

Throughout this thesis, several aspects of association rule mining will be explored. We will investigate how interestingness measures can be aggregated to produce stronger rules, how we can effectively limit the search space by only mining the *targeted rules*, how we can predict if rules are actually relevant, and how various characteristics of the change history and query affect overall behavior.

3 (A) Mining Targeted Association Rules

The first papers on association rule mining were conceived of in the context of *transactional data* such as shopping carts, where the task was to uncover patterns in what customers typically bought together. However, while the number of possible shopping items is large, the number of possible patterns between the items is *very* large. In fact, the number of possible association rules grows exponentially with the number of unique artifacts, making it infeasible to assess all rules [10]. Given that α is the number of unique artifacts, the number of possible association rules is given by the function $f(\alpha) = 3^{\alpha} - 2^{\alpha+1} + 1$ [11]. Figure 3 plots the growth. With only 10 unique artifacts the number of association rules is already *57 002*. However, in a software such as the *Linux kernel* we recorded over 150 000 unique changes[1], making the total number of possible association rules grow to the unimaginable number of 10^{71568}. It is quite clear that some sort of filtering is required for any but the smallest data sets.

[1]This number approximates the number of unique methods in the kernel

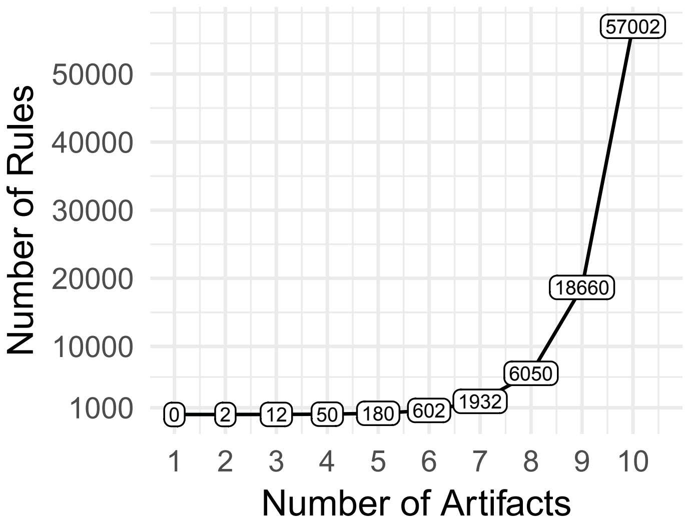


Fig. 3: The number of possible association rules as a function of the number of unique artifacts

Rather than mining *"all rules"*, initial methods focused on mining *"all frequent rules"*. Frequent rule mining was first conceived in the *Apriori* algorithm. The Apriori algorithm defines a procedure for identifying all frequent rules through the *downward closure property*, where the user defines what is "frequent" through a set threshold [10]. Here the algorithm simply uses the fact that if a rule $A \to B$ is not frequent, neither can any extension, for example $(A \cup x) \to B$. Thus a large number of rules can be pruned from the search space.

While Apriori was a significant step in making association rule mining practical, it still suffered some limitations. First, the very concept of a user supplied threshold makes it somewhat arbitrary which rules are returned. Second, a frequency threshold also implies that "low frequency" rules are not interesting, which completely dismisses recently added artifacts. Third, if any large frequent rules exist, the problem is still intractable; given a rule $A \to B$ with a number of artifacts $|A \cup B| = N$, the number of subsets of $A \cup B$ that needs to be considered before arriving at $A \to B$ is equal to $2^N - 1$.

In the first paper of this thesis (paper A, p.25) *"Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis"*, we present a mining algorithm where no frequency threshold is required as the returned rules are constrained to only those that are relevant for a given task. the algorithm TARMAQ (**T**argeted **A**ssociation **R**ule **M**ining for **A**ll **Q**ueries) is a *targeted association rule mining algorithm* especially suited for tasks such as *Change Impact Analysis* (CIA) and *change recommendation*. The key idea behind targeted rule mining, as first introduced by Srikant et al. [12], is to put constraints upon which rules should be mined. A constraint might be that only rules with a single artifact in the *consequent* (right hand side) should be mined, or perhaps that only certain artifacts are allowed to occur in either the antecedent (left hand side) or consequent. TARMAQ takes this one step further by defining a *dynamic constraint*. When conceiving TARMAQ, we took inspiration from the algorithm ROSE which was introduced by Zimmerman et al. in their seminal paper "Mining version histories to guide software changes" [6]. ROSE limits the mined rules to only those where the antecedent is equal to the *change scenario* at hand (i.e., the query). For example, if a developer has changed the methods $a()$, $b()$ and $c()$, ROSE only mine rules on the form $a(), b(), c() \rightarrow X$, where X in any single artifact which has previously changed with $a()$, $b()$ and $c()$. Unfortunately, ROSE often fails to return any rules at all. There are three patterns where this occurs: **(1)** When a new change is made, i.e., a new artifact is introduced. **(2)** When a query consists of a previously unseen combination of artifacts. **(3)** When the artifacts never has changed with anything else. TARMAQ addresses these limitations by searching for the largest subset of the query which has changed with something else in the past. We might for example find that $a()$, $b()$ and $c()$ has never changed with something else, but that $a()$, $b()$ has. In our empirical study, we find that ROSE is unable to generate recommendations for approximately 70% of queries. For the queries in those 70%, TARMAQ is able to generate highly relevant recommendations 65% of the time.[2]

4 (B) Aggregating Association Rules

Given two association rules $A \rightarrow X$ and $A \rightarrow Y$, which one is the most relevant/interesting/important? This is the problem that is addressed by weighting association rules using *interestingness measures*. We saw previously in Example 1 that *open_connection() not always* is changed when *close_connection()* is. In fact, *open_connection()* is changed 2 out of 3 times when *close_connection()* is changed, which is exactly the conditional likelihood

$$P(open\_connection()|close\_connection())$$

which can be calculated as follows if we consider the transactions ids where the two methods changed:

$$\begin{aligned}\frac{|close\_connection()_{ids} \cap open\_connection()_{ids}|}{|close\_connection()_{ids}|} &= \frac{|\{100, 104, 105\} \cap \{100, 104\}|}{|\{100, 104, 105\}|} \\ &= \frac{|\{100, 104\}|}{|\{100, 104, 105\}|} \\ &= \frac{2}{3}\end{aligned}$$

[2] 65% of the queries resulted in recommendations which had a correct artifact in their top 10

This is exactly how the *confidence interestingness measure* of the association rule $close\_connection \rightarrow open\_connection()$ is calculated. In general the confidence of a rule $A \rightarrow B$ is defined as:

$$\text{confidence}(A \rightarrow B) = P(B|A) = \frac{P(A \cap B)}{P(A)}$$

The *confidence* measure was part of the first two interestingness measures that were introduced in the field, the other being the *support* measure. The *support* of an association rule simply expresses how common it is that the artifacts in question change together, with respect to the full change history. From a probabilistic perspective, the *support* of a rule $A \rightarrow B$ is thus given by

$$\text{support}(A \rightarrow B) = P(A \cap B)$$

In addition to the *support* and *confidence*, over 40 other interestingness measures has also been presented in the literature [8, 9]. Our hypothesis in paper B *"Improving Change Recommendation using Aggregated Association Rules"* is that there exists an orthogonal technique which can be used to improve the interestingness measures given to certain association rules. The basic idea is to interpret an association rule as a piece of *evidence* that the consequent is relevant. If we for example have two rules $A \rightarrow X$ and $B \rightarrow X$, the collective evidence that X is relevant is higher than if we only had one of these rules. Consider Example 2.

Example 2. *Consider the following (historic) sequence of transactions:*

$$\mathcal{T} = [\{a, x\}, \{b, y\}, \{c, y\}, \{d, y\}, \{a, x\}]$$

*Consider further that a developer changes the following artifacts $Q = \{a, b, c, d\}$. A recommendation for Q given $\mathcal{T}$, could then realistically be as follows (*support *of each rule given in parentheses):*

$$\begin{aligned} a &\rightarrow x \quad (40\%) \\ b &\rightarrow y \quad (20\%) \\ c &\rightarrow y \quad (20\%) \\ d &\rightarrow y \quad (20\%) \end{aligned}$$

Notice how the artifact x is recommended higher than the artifact y, even though y has changed on more occasions with something in Q. However, if we aggregate the support *for y and x by simply summing the* support *values, the situation changes:*

$$\begin{aligned} b, c, d &\rightarrow y \quad (80) \\ a &\rightarrow x \quad (40) \end{aligned}$$

The order of the artifacts now more closely matches our intuition.

In our empirical evaluation we study the effect of *association rule aggregation* on 40 interestingness measures and 2 mining algorithms using change histories from 17 software systems. Here we found that aggregation had a positive small to large significant effect in all but two cases.

5 (C) Configuring the Recommendation Engine

A well known effect of the continued evolution of a software system is the increasing disorder or entropy in the system: as a result of repeated changes, the number and complexity of dependencies between parts of the code grows, making it increasingly difficult for developers to foresee and reason about the effects of changes they make to a system. The evolution is reflected in both increased number of evolutionary couplings, and increased number of *complex* couplings. As a result, parameters for rule mining may need adjustment to cope with the changing environment. In paper C "*Practical Guidelines for Change Recommendation using Association Rule Mining*" we explore the effects of both adjustable parameters as well as the effect of different change scenarios for change recommendations.

Impact of interestingness measure

Our first research question concerns which interestingness measure should be used to rank the association rules underlying a change recommendation. To this end we rank 1.2 million rule sets using 39 different interestingness measures resulting in 46.8 million change recommendations. We find a cluster of 11 interestingness measures that consistently achieve a higher precision across all included software systems, they are as follows:

casual confidence	klosgen	example and counterexample rate
collective strength	loevinger	difference of confidence
leverage	support	descriptive confirmed confidence
confidence	added value	

Impact of History Filtering

When a feature is to be implemented, or a bug is to be resolved, the number of files or methods that needs to be touched may vary greatly, which in turn results in a large variation of transactions sizes in the change history. Larger transactions may also be due to batch license updates or refactoring. In related work, transactions crossing certain size thresholds are typically filtered out to minimize noise [6, 13–15]. For our second research question we study the effect that such transaction filtering has on the precision of change recommendations. What we find is that a stricter transaction filtering than previously assumed has a positive impact on precision. In our selection of software systems, we found that an optimal filtering ranged between 4 and 10, i.e., including larger transactions would negatively impact precision.

Impact of Change Set

Developers may ask for change recommendations at any time during an implementations tasks. Furthermore, the scope of the task is unknown to the recommendation engine. For example, the developer might query after making modifications to 3 files, while in reality 3 other files must also be modified to complete the task. At a later stage the developer has modified 2 of the recommended files and issues a new query. For our third research question we hypothesize that the first query is *harder* than the latter, and we find that this assumption indeed holds in practice. In summary our findings are as follows:

The larger the task, the harder the prediction:
Generating a recommendation for a task requiring modifications to 2 files is more difficult than generating a recommendation for a task requiring modifications to 10 files. However this also depends on:

With more information, the easier the prediction:
Given a task requiring modifications to 10 files, the recommendation will become stronger as more of the required files are changed.

Note that the first finding concerns the overall size of the implementation task (how many artifacts needs to be changed), while the second finding concerns progress on the task (how many artifacts has been changed so far). Thus, a recommendations where a query has been issued after a single artifact modification, and only a single artifact is missing to complete the task, consistently exhibits the highest precision among the possible scenarios. The interaction between task size and task progress is further visualized in Table 1. Here, recommendation difficulty is given on a scale from 10 to 1, with 1 being the hardest.

Table 1: Relative impact of *task progress* and *missing changes* and their interactions. The colors indicate the task size, as shown in the legend on the right.

missing	*task progress*				*task*
changes	1	2	3	4	*size*
1	10	9	8	5	2
2	5	5	4		3
3	2	3			4
4	1				5

6 (D) The Impact of Change History Length and Age

In the same manner as paper C, paper D continues the exploration of *tunable parameters* of the recommender system in Figure 2. Concretely, the paper is dedicated to exploring the effects of *history length* and *age* on recommendations. History length here simply refers to the number of transactions that should be considered when mining association rules, while history age refers to the number of transactions that have occurred since rules were last mined from the change history. To provide intuition for how varying length and age reflects in the chosen transactions, we provide the following example:

Example 3 (History length and age). *Consider the following change history, where t_x are transactions and t_1 is the newest and t_6 the oldest.*

$$\mathcal{T} = [t_1, t_2, t_3, t_4, t_5, t_6]$$

Given some new query Q *we must select which of the transaction in $\mathcal{T}$ we should consider when generating a recommendation for* Q. *In the following table we have given some examples of different settings of history length and age, and the resulting transactions given $\mathcal{T}$:*

History Length	*History Age*	*Transactions*
1	*0*	$[t_1]$
1	*1*	$[t_2]$
2	*2*	$[t_3, t_4]$
3	*3*	$[t_4, t_5, t_6]$
6	*0*	$[t_1, t_2, t_3, t_4, t_5, t_6]$

Summary of findings on history age

An advantage of *targeted association rule mining* is that association rules are always up-to-date, i.e., they encapsulate all the available information at the time of generation. Compare this against a non-targeted approach where "all rules" are generated a priori, and then queried as needed. As it may be computationally expensive to generate the rules in the latter approach, one may not be able to regenerate rules immediately as new evidence/transactions are added to the change history. In our study of *history age* we investigate the effect of not using all available recent transactions for recommendations. In short we found overwhelming evidence that such behavior is very detrimental to the quality of recommendations; missing a single transaction has a significant effect on precision, missing 100 transactions results in a 11.5% average loss of recommendation precision. As a reference, there are on average 180 commits added to the change history of the *Linux kernel* every day.

Summary of findings on history length

Given that recent transactions are "good", are the old "bad"? Should we actively filter out transactions that reach far back into the history? Imagine that you have changed the method $M()$, if the most recent transaction also changed $M()$ you can be pretty confident that there are highly related artifacts in that same transaction. However, if the most recent transaction is not related to your change, neither the next one nor the next one, when do you stop searching? If you find that $M()$ changed 100 transactions ago, or maybe 10 000 or even 100 000 transactions ago, would you use that information? Your gut feeling perhaps tells you "no", as those transactions must be "outdated". However, we found quite the opposite. Through several large scale studies, including the *Linux* change history consisting of over 500 000 transactions, we found overwhelming evidence that older transactions do not deteriorate precision. In fact, comparing recommendations for *Linux* using 30 000 and 500 000 transactions, we found that the recommendations using the full 500 000 transactions are significantly more precise, with a *large* effect size (Cliff'd delta 0.92). In essence our findings shows that the much researched *code decay* [16] which concerns that code needs to be updated with regards to its environment, does not directly translate to a *change history decay*. Thus internal coupling between artifacts are either maintained through time, or artifacts are re-factored and renamed, which automatically filters out now un-relevant transactions.

7 (E) Predicting Relevant Recommendations

We have previously seen how a set of association rules can be sorted on their interestingness measure values to produce a change recommendation. If the interestingness measure "does its job", more relevant rules will be ranked higher than less relevant rules. But what if no rules are relevant? Or if the interestingness measure does not "do its job" and ranks a series of un-relevant rules highly? Clearly such scenarios are undesirable as it may send developers on a wild goose chase; looking for relevant code where there is none. In paper E, "*Predicting Relevance of Change Recommendations*" we hypothesize that a *history aware* approach, which considers earlier change recommendations to assess the relevance of a current recomendation, will be able to effectively filter (reduce) the number of false positives. Our approach consists of training a *random forest classification model* on previous change recommendations with *known relevance*. The model can then be used to give a *single likelihood estimate* of the relevance of new change recommendations. Incidentally, this also enables automatic assessment of recommendation relevance, thus freeing developers from having to perform such an assessment. Our work therefore facilitates tooling which can automate the change recommendation process; only notifying developers when relevant recommendations are available. Furthermore, our approach is independent of the mining algorithm which generated the recommendation, and can easily be extended to encompass any interestingness measure.

Our classification model is built using 12 features spanning characteristics of the query, change history and the resulting change recommendation. After initial training, the classification model is able provide likelihood scores for whether new (unseen) recommendations are relevant or not. Thus by varying the *threshold* for when a recommendation should be considered relevant (positive result), one can find an acceptable tradeoff between true/false positives and true/false negatives. For example, in our empirical study we found that the trained models were able to achieve 99% precision and 27% recall with a relevance threshold of 0.9.

8 Granularity of Change Histories

While the underlying data used in all five papers are based on change histories, the *granularity* varies between papers. On the coarsest level, a change history can simply consist of the files that were modified in each commit, while a very fine change history would consist of the *lines* that were modified in each file. A change history consisting only of file changes has the advantage of being completely *language agnostic*, while all finer-grained alternatives must implement some sort of language specific parsing.

In this thesis two levels of granularity is primarily explored. In paper A and D a *file granularity* is used, while the other three uses a mix of *file and method granularity*. Moreover, paper B also explores two variations on these. Example 4 goes into detail on exactly how these two different levels of granularity are extracted from the same source change sets.

Example 4 (History Parsing). *Consider the following changes to the files `A.c`, `B.cpp` and `C.yaml`, which were all added to the same commit. A '+' indicates that a line was added, while a '−' indicates that a line was deleted. Typically, when an addition and deletion comes in a pair, this indicates a* changed line. *First, in the C file `A.c`, a line was changed in the method `m1(int p)`. Secondly, in the C++ file `B.cpp`, a line was changed in the method `m2(int p)` and a public variable was changed in the parent class. Lastly, in the configuration file `C.yaml`, a single line was changed.*

A.c

```
  int m1(int p) {
-   old_line
+   new_line
  }
```

B.cpp

```
  class Class1 {
-   public var_old
+   public var_new

    int m2(int p) {
-     old_line
+     new_line
    }
  }
```

C.yaml

```
-   old_config: X
+   new_config: Y
```

To convert the above change set into a file level transaction we simply include all changed files like so: {`A.c`, `B.cpp`, `C.yaml`}. For a more fine grained transaction, parsing is required. We utilize SrcML which as of now supports `C++,C,C# and Java` *[17]. For each supported file we extract* method changes, *and also record changes outside of methods as* residual changes. *For non supported languages we simply record that the file was changed. The above change set therefore converts to the following fine grained transaction: {*`A.c:m1`*,* `B.cpp:m2`*,* `B.cpp:@residuals`*,* `C.yaml`*}*
Note that class and parameter information also is encoded on artifacts to deal with name overloading, but this information is left out of the example above for readability.

Relation between granularity and precision

With the exception of paper E, all other papers utilize the *Average Precision* (AP) measure to evaluate recommendations. In short, AP rewards ranking *correct artifacts* in the top of the recommendation, i.e., order matters as opposed to other popular measures such as *precision*. With this in mind, an observant reader might notice a drop in reported AP values when moving from the file based change histories of paper A and D, to the method based change histories of paper B and C. Thus it may appear as the quality of recommendations degrades when fine grained change histories are used as the basis for generating recommendations. However, AP values between the two granularities should not be directly compared. In a file context, recommending a correct file is *always* rewarded, while in a method context, recommending a correct file is *only* rewarded if the correct method is also recommended. While not addressed in this thesis, we envision that *granularity aware* evaluation measures can be derived to provide better precision scores in these cases. For example, if a generated recommendation is expected to

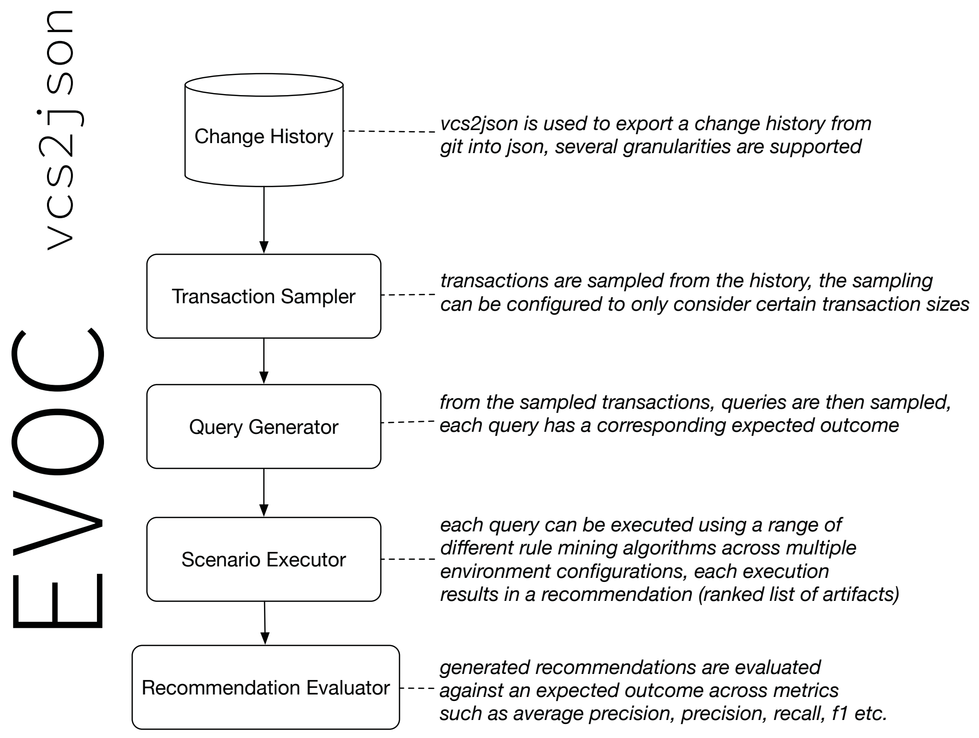

EVOC
vcs2json
Change History
vcs2json is used to export a change history from git into json, several granularities are supported
Transaction Sampler
transactions are sampled from the history, the sampling can be configured to only consider certain transaction sizes
Query Generator
from the sampled transactions, queries are then sampled, each query has a corresponding expected outcome
Scenario Executor
each query can be executed using a range of different rule mining algorithms across multiple environment configurations, each execution results in a recommendation (ranked list of artifacts)
Recommendation Evaluator
generated recommendations are evaluated against an expected outcome across metrics such as average precision, precision, recall, f1 etc.

Fig. 4

contain `fileA:methodB()`, it should be partly rewarded for recommending `fileA:methodC()`.

9 Experiment Infrastructure: EVOC

Throughout the thesis all presented hypothesis are empirically tested. To facilitate these empirical studies we developed the tools `EVOC` (**EVO**lutionary **C**oupling) and `vcs2json` (Version **C**ontrol **S**ystem 2 JSON). `vcs2json` is a command line tool written in *Ruby* for converting the output of `git log` into machine readable json. `vcs2json` is able to extract the methods that were changed in a file by comparing the Abstract Syntax Tree (AST) of the original and modified file. The AST is generated using the tool `SrcML` [17]. Relating back to Figure 1, `vcs2json` functions as our *Data Collection Mechanism*. Complementing `vcs2json` we have `EVOC`. `EVOC` is a command line tool also written in *Ruby* which is able to operate on the json change histories generated by `vcs2json`. Furthermore, it contains implementations of all association rule mining algorithms and interestingness measures used throughout the thesis. While primarily intended to be driven programmatically for experiments, `EVOC` can also be driven from the command line for single queries, and can therefore function as the *Recommendation Engine* and *User Interface* of Figure 1.

The basic idea in all studies is to emulate *change scenarios* given by a *query* and an *expected outcome*. Conceptually, a query Q represents a set of artifacts that a developer changed since the last synchronization with the version control system. In order to generate queries we simple sample a subset from a transaction T, this subset (i.e., the query) emulates a developer errantly forgetting to update the remaining artifacts of T. Thus for each query, we can easily calculate the *expected outcome* as $E = T - Q$. In this way, we can evaluate the ability of a recommendation engine to infer E from Q given the source change history. `EVOC` facilitates running millions of change scenarios in a convenient manner, while also supporting experiment critical tasks such as random sampling. Figure 4 depicts our experiment architecture with descriptions of each phase. Both `vcs2json` and `EVOC` can be installed using the `gem` command which should come with all recent Ruby installations, e.g., `gem install vcs2json`.

10 Conclusion

Change recommendation aims to improve the development process by inferring the implementation task from current changes. These recommendations may in the least reduce the time spent by developers on code navigation, and may in some cases save hours of debugging at a later stage. In this thesis the focus has been on utilizing a particular kind of source data, namely the *change history* of software systems. Within the change histories it is possible through *association rule mining* to identify relevant patterns for a set of current changes, these patterns make up the change recommendation. Papers A through D address various approaches for increasing the accuracy of change recommendations, while E presents an approach for predicting whether a recommendation *is* accurate/relevant, it thus complements the initial work.

The approach to change recommendation explored in this thesis differentiates from more traditional change recommendation approaches based on static or dynamic analysis [18–25]. In the case of static analysis, the potentially affected artifacts of a change are identified *statically*, i.e., without executing code, while dynamic analysis evaluates impact by executing the software. While static analysis typically is considered *safe* in the sense that all potentially affected artifacts are found, such analysis often *overapproximates* the impact by recommending artifacts which also are not actually affected [26]. Dynamic analysis on the other hand may *under specify* the impact by only recommending artifacts related to a specific execution of the program. Both approaches also suffer much overhead due to having to re-execute as the software evolves [24]. Furthermore, static and dynamic analysis are typically restricted to one programming language at a time, meaning that dependencies across languages are not considered for change recommendations [27]. Change recommendation based on evolutionary coupling do not suffer these limitations. However, accuracy of recommendations directly depend on the quality of the change history; if developers add unrelated artifacts to the same commit, recommendations may eventually degrade. Thus, in addition to the future work suggested by each paper, we believe an overarching task is to improve *data quality*, Possible directions may involve: **(1)** developer surveys concerning their commit habits **(2)** commit message analysis to mark commits which may contain multi-task (unrelated) artifacts **(3)** updating the change history based on artifact renaming and deletion **(4)** continued work on multi-language support for fine grained change history extraction. Furthermore, it should be of great interest to empirically study the differences between change recommendations based on static/dynamic analysis and those based on evolutionary coupling. By comparing impact sets we can calculate intersections and complements which can then be used to direct future research in either approach. Integrative approaches to change recommendation using information from both static/dynamic analysis and evolutionary coupling should also be a worthwhile undertaking.

The research effort presented in this thesis offers approaches for improving the accuracy and practical viability of evolutionary coupling based change recommendations. Such recommendations are inherently language agnostic if based on file level change granularity, only requiring that the software is version controlled. Thus industry adoption of these change recommendations is likely due to *low overhead*. The *accuracy* of recommendations can be further improved by adding fine grained language support, which enables heterogeneous couplings without further effort.

References

[1] M. J. Rochkind, "The source code control system," *IEEE Transactions on Software Engineering*, vol. SE-1, no. 4, pp. 364–370, dec 1975. [Online]. Available: http://ieeexplore.ieee.org/document/6312866/

[2] M. Robillard, R. Walker, and T. Zimmermann, "Recommendation Systems for Software Engineering," *IEEE Software*, vol. 27, no. 4, pp. 80–86, jul 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5235134http://ieeexplore.ieee.org/document/5235134/

[3] A. Mockus and J. D. Herbsleb, "Expertise browser," in *Proceedings of the 24th international conference on Software engineering - ICSE '02*. New York, New York, USA: ACM Press, 2002, p. 503. [Online]. Available: http://portal.acm.org/citation.cfm?doid=581339.581401

[4] A. Ankolekar, K. Sycara, J. Herbsleb, R. Kraut, and C. Welty, "Supporting online problem-solving communities with the semantic web," in

Proceedings of the 15th international conference on World Wide Web - WWW '06. New York, New York, USA: ACM Press, 2006, p. 575. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1135777.1135862

[5] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceeding of the 28th international conference on Software engineering - ICSE '06*. New York, New York, USA: ACM Press, 2006, p. 452. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1134285.1134349

[6] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1463228

[7] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD International Conference on Management of Data*. ACM, 1993, pp. 207–216. [Online]. Available: http://portal.acm.org/citation.cfm?doid=170035.170072

[8] L. Geng and H. J. Hamilton, "Interestingness measures for data mining," *ACM Computing Surveys*, vol. 38, no. 3, sep 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1132960.1132963

[9] K. McGarry, "A survey of interestingness measures for knowledge discovery," *The Knowledge Engineering Review*, vol. 20, no. 01, p. 39, 2005.

[10] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in *International Conference on Very Large Data Bases (VLDB)*, 1994, pp. 487–499.

[11] J. Azé and Y. Kodratoff, "Evaluation de la résistance au bruit de quelques mesures d'extraction de règles d'association." in *Extraction et Gestion des Connaissances (EGC)*, vol. 1, no. 4. Hermes Science Publications, 2002, pp. 143–154. [Online]. Available: http://dblp.uni-trier.de/rec/bib/conf/f-egc/AzeK02

[12] R. Srikant, Q. Vu, and R. Agrawal, "Mining Association Rules with Item Constraints," in *International Conference on Knowledge Discovery and Data Mining (KDD)*. AASI, 1997, pp. 67–73.

[13] A. T. T. Ying, G. Murphy, R. T. Ng, and M. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1324645

[14] A. Alali, "An Empirical Characterization of Commits in Software Repositories," Ms.c, Kent State University, 2008.

[15] H. Kagdi, M. Gethers, and D. Poshyvanyk, "Integrating conceptual and logical couplings for change impact analysis in software," *Empirical Software Engineering*, vol. 18, no. 5, pp. 933–969, oct 2013. [Online]. Available: http://link.springer.com/10.1007/s10664-012-9233-9

[16] S. Eick, T. L. Graves, A. Karr, J. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 1–12, 2001. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=895984

[17] M. L. Collard, M. J. Decker, and J. I. Maletic, "srcML: An Infrastructure for the Exploration, Analysis, and Manipulation of Source Code: A Tool Demonstration," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, sep 2013, pp. 516–519. [Online]. Available: http://ieeexplore.ieee.org/document/6676946/

[18] B. Li, X. Sun, H. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," *Software Testing, Verification and Reliability*, vol. 23, no. 8, pp. 613–646, dec 2013. [Online]. Available: http://doi.wiley.com/10.1002/stvr.1475

[19] L. Badri, M. Badri, and D. St-Yves, "Supporting predictive change impact analysis: a control call graph based technique," in *12th Asia-Pacific Software Engineering Conference (APSEC'05)*. IEEE, 2005, p. 9 pp. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1607149http://ieeexplore.ieee.org/document/1607149/

[20] L. Huang and Y.-T. Song, "Precise Dynamic Impact Analysis with Dependency Analysis for Object-oriented Programs," in *5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007)*. IEEE, aug 2007, pp. 374–384. [Online]. Available: http://ieeexplore.ieee.org/document/4721315/http://ieeexplore.ieee.org/document/4296961/

[21] L. Hattori, D. Guerrero, J. Figueiredo, J. Brunet, and J. Dam, "On the Precision and Accuracy of Impact Analysis Techniques," in *Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*. IEEE, may 2008, pp. 513–518. [Online]. Available: http://ieeexplore.ieee.org/document/4529870/

[22] M. Petrenko and V. Rajlich, "Variable granularity for improving precision of impact analysis," in *2009 IEEE 17th International Conference*

on Program Comprehension. IEEE, may 2009, pp. 10–19. [Online]. Available: http://ieeexplore.ieee.org/document/5090023/

[23] X. Sun, B. Li, C. Tao, W. Wen, and S. Zhang, "Change Impact Analysis Based on a Taxonomy of Change Types," in *Computer Software and Applications Conference (COMPSAC)*. IEEE, jul 2010, pp. 373–382. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5676283

[24] J. Law and G. Rothermel, "Whole Program Path-Based Dynamic Impact Analysis," in *International Conference on Software Engineering (ICSE)*. IEEE, 2003, pp. 308–318. [Online]. Available: http://dl.acm.org/citation.cfm?id=776816.776854

[25] T. Apiwattanapong, A. Orso, and M. J. Harrold, "Efficient and precise dynamic impact analysis using execute-after sequences," in *Proceedings of the 27th international conference on Software engineering - ICSE '05*. New York, New York, USA: ACM Press, 2005, p. 432. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1062455.1062534

[26] D. W. Binkley, N. Gold, M. Harman, S. Islam, J. Krinke, and S. Yoo, "ORBS and the Limits of Static Slicing," in *International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2015, pp. 1–10. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2635868.2635893

[27] A. R. Yazdanshenas and L. Moonen, "Crossing the boundaries while analyzing heterogeneous component-based software systems," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011, pp. 193–202. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2011.6080786http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6080786

Part II

Papers

Paper A

Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis

Thomas Rolfsnes, Stefano Di Alesio, Razieh Behjati,
Leon Moonen and Dave W. Binkley

Accepted for publication in the main research track of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER), March 14-18, 2016.

© 2016 IEEE
The layout has been revised.

Abstract

Software change impact analysis aims to find artifacts potentially affected by a change. Typical approaches apply language-specific static or dynamic dependence analysis, and are thus restricted to homogeneous systems. This restriction is a major drawback given today's increasingly heterogeneous software. Evolutionary coupling has been proposed as a language-agnostic alternative that mines relations between source-code entities from the system's change history. Unfortunately, existing evolutionary coupling based techniques fall short. For example, using Singular Value Decomposition (SVD) quickly becomes computationally expensive. An efficient alternative applies targeted association rule mining, but the most widely known approach (ROSE) has restricted applicability: experiments on two large industrial systems, and four large open source systems, show that ROSE *can only identify dependencies about 25% of the time.*

To overcome this limitation, we introduce TARMAQ*, a new algorithm for mining evolutionary coupling. Empirically evaluated on the same six systems,* TARMAQ *performs consistently better than* ROSE *and* SVD*, is applicable 100% of the time, and runs orders of magnitude faster than* SVD*. We conclude that the proposed algorithm is a significant step forward towards achieving robust change impact analysis for heterogeneous systems.*

1 Introduction

As a software system evolves, the amount and complexity of interactions in the code grows. For a developer, it therefore becomes increasingly challenging to be in control of the impact of a change made to the system. One potential solution to this problem, change impact analysis [1–4], aims to find artifacts (e.g., files, methods, classes etc.) affected by a given change. This knowledge can then be used either as direct feedback to the developer, or as the basis for another down-stream task such as test-case selection and prioritization.

Traditionally, change impact analysis has been performed using static or dynamic dependence analysis (e.g., by identifying the methods that call a changed method). One advantage of such approaches is that they are considered *safe*, as all potentially affected artifacts are found [5]. However, in recent years there has been an investigation of alternative approaches. This search is motivated, in part, by limitations in existing techniques. For example, static and dynamic dependence analysis are generally language-specific, making them unsuitable for the analysis of heterogeneous software systems [6]. In addition, they can involve considerable overhead (e.g., dynamic analysis' need for code-instrumentation), and tend to over-approximate the impact of a change.

One alternative is to identify dependencies through *evolutionary coupling*. Such couplings differ from the ones found through static and dynamic dependence analysis, in that they are based on *how* the software system has evolved over time. Using this information in essence attempts to tap into the developer's inherent knowledge of the inner workings of the system. This knowledge can manifest itself in several ways, for example through commit-comments, bug-reports, context-switches in an IDE etc. In this paper we consider *co-change* as a basis for establishing evolutionary couplings. Co-change information can be extracted from a project's version control system, its issue tracking system, or both, depending on how the project maintains its software revision history.

The goal of evolutionary coupling is to mine connections between entities in the software from the co-change data. While several levels of granularity are possible, in this paper we focus on connections at the file level. Observe that this is without loss of generality, as the mining algorithms are agnostic to the choice of granularity. Provided that suitably fine-grained co-change data is obtained, the algorithms will just as well relate methods or variables as files in a system.

There are several options to mine evolutionary coupling from co-change data. For example, although it is computationally expensive, Singular Value Decomposition (SVD) can be used to form clusters of files. These clusters can then be used to predict files likely to change together [7]. A considerably faster approach is based on Srikant et al.'s *targeted association rule mining* algorithm [8].

However, the off-the-shelf use of this algorithm has a significant limitation causing a lack of *applicability* (a notion formalized in section 4). Consider ROSE, which uses the algorithm for software change recommendations [9]. Experiments with ROSE on six large systems show that it can only find evolutionary couplings about 25% of the time.

This paper presents TARMAQ, a new generalized algorithm for targeted association rule mining that overcomes this limitation. Moreover, we make the following additional contributions: we classify the limitations of existing algorithms, and we empirically evaluate the performance of TARMAQ on two industrial software systems and four open source software systems. We find that TARMAQ performs consistently better than two popular alternatives, is applicable 100% of the time, and runs orders of magnitude faster than SVD. We conclude that the new algorithm is a significant step forward towards achieving robust change impact analysis for heterogeneous systems.

2 Research Questions

Our research is driven by a desire to conduct impact analysis on large heterogeneous systems via evolutionary coupling. Despite its limited applicability, targeted association rule mining has shown promise in addressing software engineering problems. Combined, these two observations lead us to investigate the following research questions:

RQ1 What are the limitations of existing techniques to analyse evolutionary coupling?

RQ2 Can we devise an alternative technique that does not suffer from these limitations?

RQ3 How well does the best alternative perform in comparison to the state-of-the-art?

The remainder of the paper is organized as follows: section 3 defines some terminology and provides background on the use of association rule mining. The limitations of existing algorithms (RQ1) are discussed in section 4. In section 5 we address RQ2 by introducing TARMAQ, a new algorithm for targeted association rule mining algorithm in software engineering context. We address RQ3 by empirically evaluating TARMAQ and state-of-the-art algorithms on a series of large software systems in Sections 6 – 8. We discuss related work in section 9, and we conclude in section 10.

3 Background

Agrawal et al. introduced the concept of *association rule mining* as the discipline aimed at inferring relations between *entities* of a dataset [10]. The relations, called *association rules*, are identified from a collection of transactions where each transaction is a subset of the entities. For example, consider the classic application of analyzing shopping cart data: if multiple transactions include bread and butter then the corresponding association rule would be *bread* $\rightarrow$ *butter*. This can be read as *"if you buy bread, then you are also likely to buy butter"*.

In the context of evolutionary coupling for software systems, the entities are the files of the system and the collection of transactions is the change history $\mathcal{H}$ of the system. Note that, in general, entities in the system can be considered at other levels of granularity, such as method- or procedure-level. Each transaction $T \in \mathcal{H}$ is a *commit* of changed files, i.e., a transaction includes the set of files that were either changed or added while addressing a given bug or feature addition (creating a *logical dependence* [11]).

Finally, an *association rule* is an implication of the form $A \rightarrow B$, where A and B are disjoint, A is referred to as the *antecedent*, and B is referred to as the *consequent*. In our use, the association rule $A \rightarrow B$ denotes that *"if the files in A change, then the files in B are also expected to change"*.

Several measures are used to reason about the rules. First, the *frequency*, denoted ϕ, of association rule $A \rightarrow B$ is the number of transactions in $\mathcal{H}$ where items in A and B change together:

$$\phi(A \rightarrow B) \stackrel{\text{def}}{=} |\{T \in \mathcal{H} : A \cup B \subseteq T\}| \tag{A.1}$$

Usually, the frequency of a rule is normalized by dividing by the number of transactions. For this purpose, the *support*, denoted σ, of a rule $A \rightarrow B$ is defined as the frequency of a rule divided by the number of transactions in the history:

$$\sigma(A \rightarrow B) \stackrel{\text{def}}{=} \frac{\phi(A \rightarrow B)}{|\mathcal{H}|} \tag{A.2}$$

Intuitively, higher support for a rule means that it is more likely to hold. Alternatively, rules with low support identify relations that rarely occur. For this reason, a minimum threshold on support is often used to filter out uninteresting rules.

The final measure used, *confidence*, denoted κ, measures the reliability of the inference made by a rule. The confidence of a rule is defined as its frequency divided by the number of transactions that contain its antecedent.

$$\kappa(A \rightarrow B) \stackrel{\text{def}}{=} \frac{\phi(A \rightarrow B)}{|\{T \in \mathcal{H} : A \subseteq T\}|} \tag{A.3}$$

Confidence measures the ratio of the transactions in which the files in A *and* B are present, to the transactions where files in A are present. Consequently, the higher the confidence of a rule $A \rightarrow B$, the higher the chance that when the files in A are modified, then the files of B are also modified.

As originally defined [10], association rule mining generates rules that express patterns in a complete data set. However, some applications can exploit a more focused set of rules. *Targeted association rule mining* [12] is a refinement that focuses the generation of rules on a particular *query* supplied by the user. It does so by removing all transactions that are not related to the query from the database of transactions (the change history $\mathcal{H}$ in our context), which results in a dramatic reduction of execution time [12].

4 Problem Description

This section characterizes a key limitation of applying off-the-shelf targeted association rule mining to the problem of providing developer change recommendations. As described earlier, the technique removes all transactions that are not related to the query from the database of transactions used for rule generation. However, this strategy can often filter away all the transactions, leaving an empty database, and rendering the technique incapable of producing a recommendation. Before considering the causes of this limitation, we formalize the notion of *applicability*:

Definition 1 (Applicability). *Given a history $\mathcal{H}$ and a query Q, a targeted association rule mining technique $\mathcal{T}$ is said to be* applicable *to query Q if the filtered history after removing all transactions not related to Q is non-empty. Conversely, we say that $\mathcal{T}$* is not applicable *when the filtered history for Q is empty.*

It turns out that there are two patterns that lead to a lack of applicability of the existing techniques. The first pattern is when a file in the query has never occurred before in any transaction of the history. The second pattern is a query of previously seen files that nonetheless have not been seen together as a proper subset of a single transaction. The second pattern requires a proper subset because a transaction that exactly matches the query involves no other files, and thus there are no other files to recommend. Therefore, nothing can be learned from such a transaction.

To illustrate the two patterns, let *a, b, c, d* and *e* represent five source files belonging to an evolving software system. Suppose that the version history of the system, $\mathcal{H}$, contains three transactions:

$$\mathcal{H} = [\{a,b,c\},\{a,d\},\{c,d\}]$$

Then, the queries $q_1 = \{e,d\}$ and $q_2 = \{a,c,d\}$ exemplify the two patterns for which existing targeted association rule mining techniques are not applicable. q_1 includes a new file e which does not occur in any transaction of $\mathcal{H}$. In this case techniques such as Rose [9] fail to provide any recommendation. In the second example, the files in q_2 have all occurred in the history, but never together. Here again Rose fails to provide a recommendation. In the following, we refer to these queries as *unseen* queries because they include a pattern that is unseen in the change history.

The opposite of an unseen query is a *seen* query. For such queries Rose is able to provide a recommendation. Two examples of seen queries are $q_3 = \{a\}$ and $q_4 = \{a,b\}$. Because there is at least one transaction in $\mathcal{H}$ that is a proper superset of these queries, Rose can produce a result.

To find out to what extent this limitation affects the applicability of Rose and similar approaches, such as that of Ying et al. [13], we studied the distribution of unseen and seen queries in four open source repositories, namely, Git[1], Apache HTTP Server[2], Linux Kernel[3], MySQL[4] as well as two software

[1] `https://www.openhub.net/p/git`
[2] `https://www.openhub.net/p/apache`
[3] `https://www.openhub.net/p/linux`
[4] `https://www.openhub.net/p/mysql`

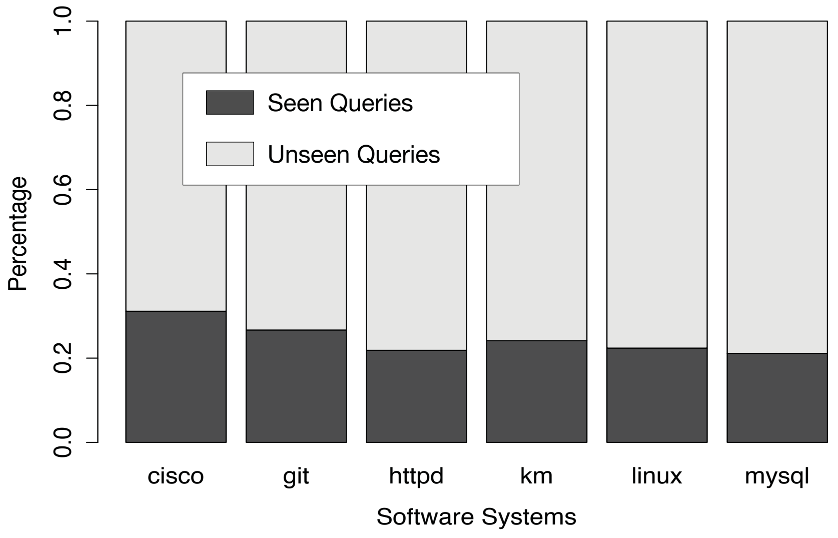


Fig. A.1: The percentage of queries that were found to be *seen* or *unseen* in the change histories of each of the software systems.

repositories from our industry partners, Cisco Norway[5] and Kongsberg Maritime (KM)[6]. The results are reported in Figure A.1 [7]. The high percentage of unseen queries in all six cases implies that traditional targeted association rule mining cannot produce results a significant amount of the time.

We can answer our first research question, RQ1, in the affirmative; it is possible to characterize the limitations of existing evolutionary coupling based techniques. Specifically, the two patterns described in this section prevent current techniques from returning a result. This is because these techniques use an over-restrictive history filtering, which only keeps transactions where *all* the files in the query are present. This suggests the need for alternative and more relaxed rule filtering methods.

5 Proposed Solution: Tarmaq

This section presents Tarmaq, a novel algorithm implementing *Targeted Association Rule Mining for All Queries*. Tarmaq takes as input a transaction history $\mathcal{H}$ and a query Q, and generates a ranked list $\mathcal{F}$ of files where higher ranked files are more related to Q. As shown in Algorithm 1, Tarmaq consists of three steps: transaction filtering, rule creation, and finally the ranked list creation.

[5] `http://www.cisco.com/web/NO/index.html`

[6] `http://www.km.kongsberg.com/`

[7]The study *re-enacted* the change history,basically using commits as a query over the change history up to the point that the commit was made.

Algorithm 1: Tarmaq

```
Require: The history: $\mathcal{H}$, and the query: $Q$
Ensure: A ranked list of files: $\mathcal{F}$
1  {Filtering Step}
2  $k \leftarrow 0$
3  $\mathcal{H}_f \leftarrow \emptyset$ {$\mathcal{H}_f$: the filtered history}
4  for all $T \in \mathcal{H}$ do
5     if $|Q \cap T| = k$ then
6        $\mathcal{H}_f \leftarrow \mathcal{H}_f \cup \{T\}$
7     else if $|Q \cap T| > k$ then
8        $k \leftarrow |Q \cap T|$
9        $\mathcal{H}_f \leftarrow \{T\}$
10 {Rule Creation Step}
11 $\mathcal{R} \leftarrow \emptyset$ {$\mathcal{R}$: the set of rules}
12 for all $T \in \mathcal{H}_f$ do
13    $Q' \leftarrow Q \cap T$
14    for all $x \in T \setminus Q'$ do
15       $\mathcal{R} \leftarrow \mathcal{R} \cup \{Q' \rightarrow x\}$
16       $update(\sigma(Q' \rightarrow x))$
17       $update(\kappa(Q' \rightarrow x))$
18 {Ranked List Creation Step}
19 $\mathcal{R}_s \leftarrow sort(\mathcal{R})$ {sorts using $\sigma$ and $\kappa$}
20 for all $i \in [1..\ length(\mathcal{R}_s)]$ do
21    $\mathcal{F}[i] \leftarrow consequent(\mathcal{R}_s[i])$
22 return $\mathcal{F}$
```

Given a query $Q = \{file_1, file_2, \dots file_n\}$, the transaction filtering extracts from $\mathcal{H}$ those transactions that have the largest intersection with Q. More formally, transaction $T \in \mathcal{H}$ is kept if $|T \cap Q| = k$, where k is the size of the largest subset of Q seen in the history. In contrast, Rose keeps only those transactions where $Q \subseteq T$. Revisiting the example from section 4, for q_2 the filtering removes none of the three transactions as each includes two of the files from q_2. However, for q_3 the final transaction, $\{c, d\}$, is removed by the filtering because $\{a\} \cap \{c, d\} = \emptyset$. After filtering, the second step is rule creation. Tarmaq generates rules of the form $Q' \rightarrow x$, where x represents a single file and Q' is a subset of Q with $|Q'| = k$. Such a rule is created if and only if there exists a transaction $T \in \mathcal{H}$ such that $Q' \cup \{x\} \subseteq T$. Considering the example from section 4, Tarmaq generates three rules for q_3: $\{a\} \rightarrow \{b\}$, $\{a\} \rightarrow \{c\}$, and $\{a\} \rightarrow \{d\}$. Note that when $\mathcal{H}$ contains at least one transaction T such that $Q \subset T$, Tarmaq generates the same set of rules as Rose. However, unlike Rose, when Q is not contained in any T, TARMAQ generates rules whose antecedent is as close to Q as possible. In contrast, ROSE fails to generate any rules. In the final step TARMAQ produces the ranked list $\mathcal{F}$ of recommended files. This is done by first sorting on the support of each rule, and in the case of ties (which are likely) on the confidence of each rule. The final list of files is then produced by mapping each rule to its consequent.

The rational for having a single file as the consequent includes both efficiency and utility. More general rules would involve Q' implying that a set of files should be included rather than a single file. Algorithms for generating such rules are computationally more expensive and may require a search through all possible subsets, which is an exponential computation and thus quickly becomes a performance concern. Furthermore, there is no loss of utility because in a software context it is sufficient to recommend files independently. The independence of software entities was exploited by ROSE, which also produces singleton consequents [9].

By looking for transactions that contain subsets of Q instead of Q itself, we obtain some recommendation evidence in the absence of transactions involving all of Q. For example, consider the situation where an engineer has modified files a, b, and c to fix a bug, but errantly forgot to modify file x. In this case, we want to mine the rule $\{a, b, c\} \rightarrow x$ with high confidence. However, this is not possible if c is new or has not been previously changed. Nevertheless, assuming that a, b, and x have frequently changed together before, TARMAQ produces the rule $\{a, b\} \rightarrow x$ with high confidence. Thus, in answer to RQ2, using the largest intersection, we can develop a technique that does not suffer from a lack of applicability.

6 Evaluation

The evaluation compares TARMAQ, ROSE, and SVD by emulating a developer's need for a change-recommendation tool. To facilitate the three experiments we assume that the files of a transaction are related and evaluate performance by partitioning transactions into a query and an expected output. To describe the evaluation we first describe the subject systems used as test subjects in the experiments. We then detail the query generation process, the query evaluation, and finally, the implementation and execution environment.

6.1 Subject Systems

To assess the algorithms in a variety of conditions, we selected six systems having varying size and frequency of commits. Two of these are systems from our industry partners, Kongsberg Maritime (KM) and Cisco Norway. KM is a leading company in the production of systems for positioning, surveying,

Table A.1: Characteristics of the evaluated software systems

Software System	Unique Files	Avg. tx. size (# files)	History covered by 10 000 transactions
MySQL	21854	10.1	2.34 years
Git	2141	1.9	4.2 years
Apache HTTP Server	7905	6.9	7.18 years
Linux Kernel	9021	2.2	0.15 years
Kongsberg Maritime	35111	5.1	15.97 years
Cisco Norway	41701	6.2	1.07 years

Software System	Languages used*
MySQL	C++ (54%), C (19%), JavaScript (17%), 23 other (10%)
Git	C (45%), shell script (35%), Perl (9%), 14 other (11%)
Apache HTTP Server	XML (56%), C (32%), Forth (8%), 19 other (4%)
Linux Kernel	C (94%), 16 other (6%)
Kongsberg Maritime	C++, C, XML, other build/config
Cisco Norway	C++, C, C#, Python, Java, XML, other build/config

* language information from `http://www.openhub.net`, percentages for the industrial systems are not disclosed.

navigation, and automation of merchant vessels and offshore installations. Cisco Norway is the Norwegian division of Cisco Systems, a worldwide leader in the production of networking equipment. We validated Tarmaq in the common software platform KM uses in applications in the maritime and energy domain, and in the Cisco software product line for professional video conferencing systems. The other four systems are part of well known open-source projects, namely Apache HTTP Server, Linux Kernel, MySQL, and Git. Table A.1 summarizes descriptive characteristics of the software systems used in the evaluation. The table shows that the systems we selected vary from medium to large size, with up to forty thousand different files committed in the transaction history. Furthermore, the oldest transactions from the system histories are fifteen years old in the case of KM. Note that all the systems are heterogeneous, i.e., they are implemented in more than a single programming language.

A key aspect of each system is the way its authors interact with the version control system. This usage typically depends on the software process adopted by the developer's organization. For example, agile teams tend to frequently commit small incremental changes to the project artifacts following the *"commit early, commit often"* philosophy. On the other hand, in more

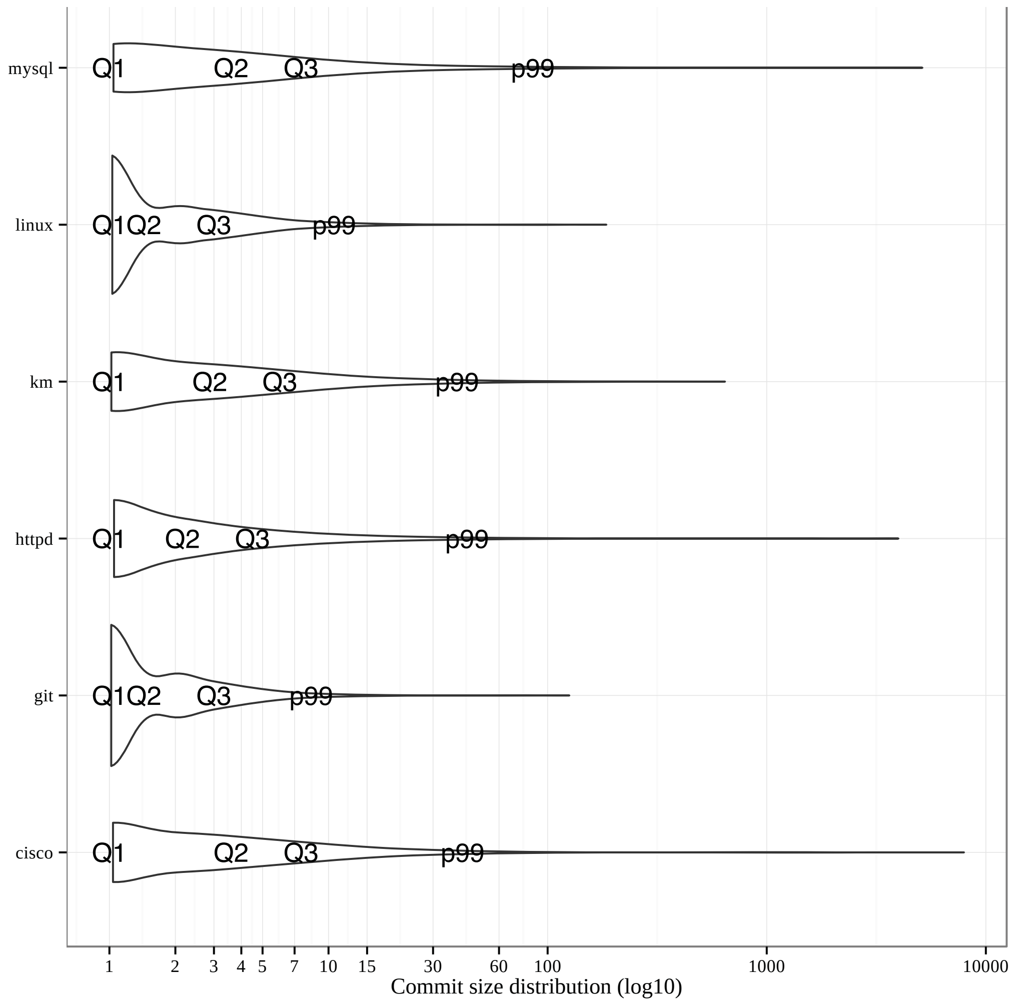


Fig. A.2: Distribution of commit sizes for the different cases

traditional software processes developers might commit only on a monthly basis, with each commit changing a large number of files. To gain some understanding of the commit patterns used, Figure A.2 shows violin plots of the six software systems. For each plot the x-axis shows commit size using a log scale. Each violin includes quartile markers, Q1, Q2, and Q3, and a marker for the 99^{th} percentile, p99. From these plots certain patterns emerge. For example, Linux and Git are dominated by small commits while MySQL and KM include considerably more larger commits. This variety of patterns is relevant to our evaluation.

6.2 Determining Transactions

While our approach is not dependent on the versioning system used, adaptors still need to be written for each versioning system. These adaptors take a change-history from a specific versioning system as input, and output a set of transactions conforming to a common format. Our prototype tool uses this common format.

Depending on the versioning system however, it might not always be perfectly clear what constitutes a "transaction", as defined as "a set of related changes"; For each versioning system, a best approximation should be made. For example, in the Concurrent Versions System (CVS), changes are not explicitly grouped into commits, and it is therefore necessary to consider timestamps (e.g., sliding time windows), log messages and developer identity to define transactions [14].

Another aspect that needs to be considered is if previous commits can be modified (i.e., history rewriting). If this is not possible in the versioning system, a developer who has forgotten to commit a relevant file needs to make an additional commit. Again, a sliding time window is a potential solution here.

For this paper, all chosen software-systems use the Git version control system.[8] Our adaptor for Git treats the files changed in one commit as one transaction. Merge commits however, are ignored through the use of the *'–no-merges'* option. Additionally it should be noted that Git supports history rewriting through the *'–amend'* option and the *'rebase'* command. These commands obviate the need for employing time windows in the Git adaptor. Finally, it should also be noted that only transactions from the *main branch* are considered.

6.3 Query Generation Process

Conceptually, a query Q represents a set of files that a developer changed since the last synchronization with the version control system. The key idea behind our evaluation is to generate, starting from a transaction T, a set of queries that emulate a developer errantly forgetting to update some subset of T.

The first step in the process is to select a set of transactions. The distribution plots are clearly skewed towards small commits. In fact, 75% of the commits have ten or fewer files while 90% have thirty or fewer files. For this reason, and because we assume that larger commits often consist of unrelated files committed together because of a directory reshuffle or license change, we follow the work of Zimmerman et al. [9] and remove transactions of more than thirty files.

From the remaining commits we sample forty commits for each transaction size between two and thirty. We use the resulting 1160 transactions to form the queries used to investigate how the three algorithms behave over a range of transaction sizes.

To mimic a developer forgetting files, we partition each of the 1160 transaction T into a non-empty query Q and a non-empty expected outcome

[8]With the exception of Kongsberg Maritime, where a special adaptor was written.

$E \overset{\text{def}}{=} T \setminus Q$. In this way, we can evaluate the ability of an algorithm to infer E from Q. To investigate a range of query sizes, we generate one query for each size from one to $|T| - 1$. For example, for a transaction of size 4, we generate three queries of size 1, 2, and 3, whose expected outcomes thus have sizes 3, 2 and 1, respectively. Note that we do not sample commits of size one because they cannot be split into a query and expected outcome.

6.4 Query Evaluation

Evaluating the generated queries requires executing each query and then comparing the resulting ranked lists. To execute each query Q that was generated from transaction T, we use the 10 000 commits prior to T as the history. In these experiments 10 000 represents a balance between to short a history, which would lack sufficient connections, and to long a history, which is inefficient and can even be misleading when previously connected files are not longer connected.

Comparing the ranked lists produced by Tarmaq, Rose, and svd requires an appropriate performance measure. Prior work on association rule mining typically uses *precision* and *recall* as performance metrics. The *precision* of a recommendation is the ratio of the number of correct items recommended to the total number of items recommended. The *recall* of a recommendation is the ratio of the number of correct items recommended to the total number of correct items.

As a practical consideration, while precision and recall are designed for unordered results a recommendation tool's output is a *ranked list*. Consider the difference between a single correct recommendation occurring at the beginning of the list and a single correct recommendation occurring further down the list. These two have the same precision and recall. However, the correct recommendation at the beginning of the list is far more valuable, because it is far more likely to be considered. This is a well known phenomenon in information retrieval where, for example, internet searchers rarely look past the first ten results [15].

A more appropriate performance measure in the context of a ranked list is the *average precision* (*AP*). For query Q producing ranked recommendation list R, *AP* is defined as

$$AP(Q,R) \overset{\text{def}}{=} \sum_{k=1}^{|R|} P(k) * \triangle r(k) \tag{A.4}$$

where $P(k)$ is the precision calculated on the first k files in the list, (i.e., the precision@k) and $\triangle r(k)$ is the *change in recall* calculated only on the $k-1^{\text{th}}$ and k^{th} files, i.e., the fractional increase in true positives compared to the previous rank. Table A.2 illustrates the computation of AP, $P(k)$, and $\triangle r(k)$ given the ranked list $[c, a, f, g, d]$ and the expected outcome $\{c, d, f\}$.

Finally to compare the performance of two tools, we use the *mean average precision* (MAP) computed over a set of queries. A tool producing a higher MAP value is, on average, producing better results. In addition, we compare the tools based on the total wall-clock time taken to execute a collection of queries.

6.5 Implementation and Execution Environment

We implemented all three algorithms in RUBY, using LAPACKE C to implement SVD through the NMatrix gem [16]. Our implementation is open-source and can be found online.[9] We performed the experiment executing the algorithms, one at a time, on a *c4.2xlarge* Amazon EC2 instance.[10]

7 Results

This section addresses RQ3 by reporting objective measures of the data from our evaluation. Our interpretation of the data can be found in section 8. The results are organized according to the following questions:

RQ3.1 What is the overall performance of all algorithms?

RQ3.2 What is the performance on seen queries of all algorithms?

RQ3.3 What is the performance on unseen queries of TARMAQ and SVD?

RQ3.4 Is there a significant difference in performance on seen and unseen?

[9] `https://bitbucket.org/evolveIT/tarma`
[10] `https://aws.amazon.com/`

Table A.2: Example of average precision calculation

Consider as relevant files: c, d, f

Rank (k)	File	$P(k)$	$\triangle r(k)$
1	c	1/1	1/3
2	a	1/2	0
3	f	2/3	1/3
4	g	2/4	0
5	d	3/5	1/3

$$AP = 1/1 * 1/3 + 1/2 * 0 + 2/3 * 1/3 + 2/4 * 0 + 3/5 * 1/3 \approx 0.75$$

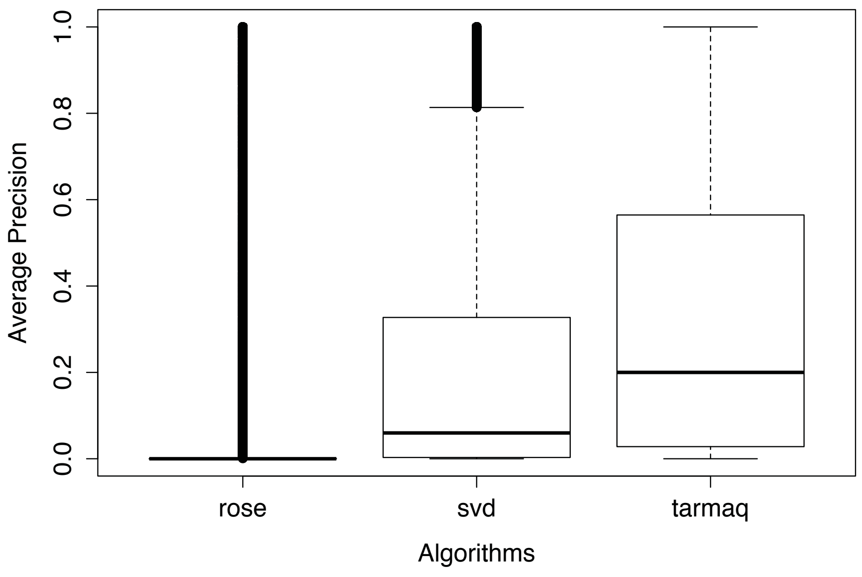


Fig. A.3: Overall distribution of average precision for each algorithm

RQ3.5 How do the algorithms perform on the individual software systems?

While not unexpected, we found that the average precision distribution for the different software systems and algorithms were not normally distributed, we therefore use the *Friedman Test*, a non-parametric test for differences between several samples.

7.1 Overall Performance on Average Precision (RQ3.1)

The overall performance implies what is to be expected of each algorithm when the type of query is unknown (seen vs unseen), which would normally be the case. We test the following hypothesis:

H_0 The average precision distribution generated by each algorithm is the same.

H_1 The average precision distribution generated by each algorithm is different.

Figure A.3 shows the overall distribution of average precision for each of the investigated algorithms. Here it is clear that Rose cannot produce results for a significant number of the queries, and therefore ends up with a high percentage of 0 AP values. Furthermore we see that Tarmaq has a higher median than svd and also a larger interquartile range.

A Friedman rank sum test[11] of the distributions yields a p-value < 0.00001, and we therefore reject H_0 and conclude that there is a significant difference between the algorithms.

Since we accept H_1, we can do a post-hoc test to actually look at which algorithms were different, to this end we use individual Wilcoxon signed rank tests[12]. Since we do multiple comparisons (tests) we have to use Bonferroni adjustment on what should be considered significant p-values. With three factor levels (three algorithms) it is sufficient with two directional tests to establish Tarmaq's place in the ordering of algorithms. The Bonferroni adjustment is given by dividing the desired alpha level by the number of performed tests, we thus get an adjusted alpha of 0.05/2 = 0.025.

We have the following hypotheses:

$H_0^{\mathbf{TvR}}$ The average precision distribution generated by Tarmaq is less than that of Rose.

$H_1^{\mathbf{TvR}}$ The average precision distribution generated by Tarmaq is greater than that of Rose.

$H_0^{\mathbf{TvS}}$ The average precision distribution generated by Tarmaq is less than that of svd.

$H_1^{\mathbf{TvS}}$ The average precision distribution generated by Tarmaq is greater than that of svd.

In Table A.3 we provide the p-values of the two Wilcoxon tests. In both cases we can safely reject the null-hypothesis, and conclude that on overall, Tarmaq performs better than both Rose and svd.

7.2 Overall Performance on Time (RQ3.1)

To investigate the execution cost of each algorithm, we perform the same analysis as we did for average precision. The overall execution time distributions can be found in Figure A.4. At once we see that svd have an

[11] We used friedman.test in R.
[12] We used wilcox.test in R with: alternative = 'greater', paired = TRUE.

Table A.3: p-values for Wilcoxon pairwise multiple comparisons

	Tarmaq	conclusion
Rose	p-value < 0.00001	reject H_0^{TvR}
svd	p-value < 0.00001	reject H_0^{TvS}

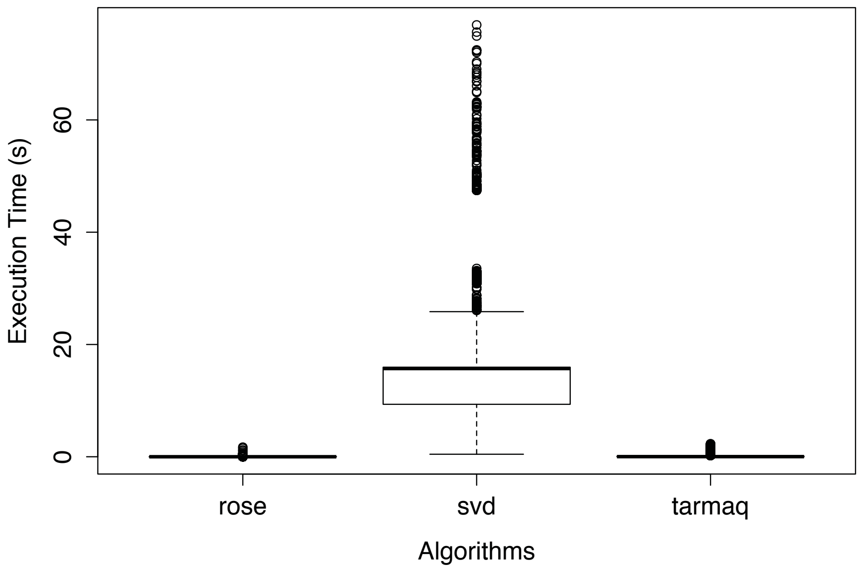


Fig. A.4: Overall distribution of execution time for each algorithm. Note that Rose executes faster because it does not produce results in all cases.

obviously larger spread in execution times than Rose and Tarmaq, and a Friedman rank sum test does indeed show significant difference between the distributions (p-value < 0.00001). The more interesting analysis is to compare Rose and Tarmaq, as they have a much closer distribution. A Wilcoxon test showed that Rose was significantly faster than Tarmaq (p-value < 0.00001), a large part of this however, might be attributed to the large number of queries where Rose does not produce results (cannot find relevant transactions), and therefore does not need to perform any rule creation. The mean execution time for Tarmaq was 0.09 s, 0.003 s for Rose, and 17.5 s for svd.

7.3 Performance on Seen Queries (RQ3.2)

In this section and in subsection 7.4 we look at algorithm performance on either seen or unseen queries respectively. This division gives us a better view into how the two types of queries differ.

In Figure A.5 we provide the distribution of average precision on all seen queries for all algorithms. We can see that, as expected, Rose and Tarmaq have equal distributions on seen queries, while svd performs worse. A Friedman test on the distributions was found significant (p-value < 0.00001), i.e., there is a difference between the algorithms. A test of significant difference between Rose and Tarmaq is unneeded, as they produce exactly the same result here, but a Wilcoxon test of Tarmaq and svd was found significant (p-value < 0.00001). We can therefore conclude that both Tarmaq and Rose

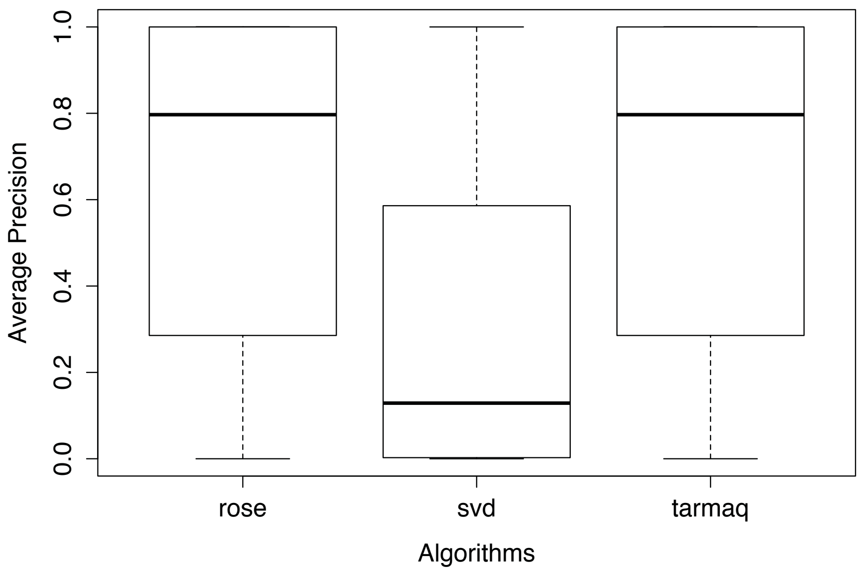


Fig. A.5: Distribution of average precision on seen queries for each algorithm

perform better than SVD on seen queries.

7.4 Performance on Unseen Queries (RQ3.3)

When a query consists of files that have never changed together before, we consider the query to be unseen. Since Rose cannot produce recommendations on unseen queries, only Tarmaq and SVD are evaluated in this section. With only two subjects, we can use the Wilcoxon test directly without Bonferroni correction.

Figure A.6 shows a boxplot of the average precision distribution of Tarmaq and SVD on unseen queries. A Wilcoxon test on Tarmaq and SVD distributions proved to be significant (p-value < 0.00001), we can therefore conclude that Tarmaq performs better than SVD on unseen queries.

7.5 Seen vs. Unseen Queries (RQ3.4)

In this section we investigate the difference in performance on seen and unseen queries, i.e., if the distributions shown in Figure A.5 and Figure A.6 significantly differ. To get a good overview we have plotted the overall distributions in Figure A.7. To compare the distributions we use an unpaired Wilcox test, as the query population obviously is different for the seen and unseen queries. The test yields a p-value < 0.00001, and we can therefore conclude that unseen queries are significantly harder than the seen queries.

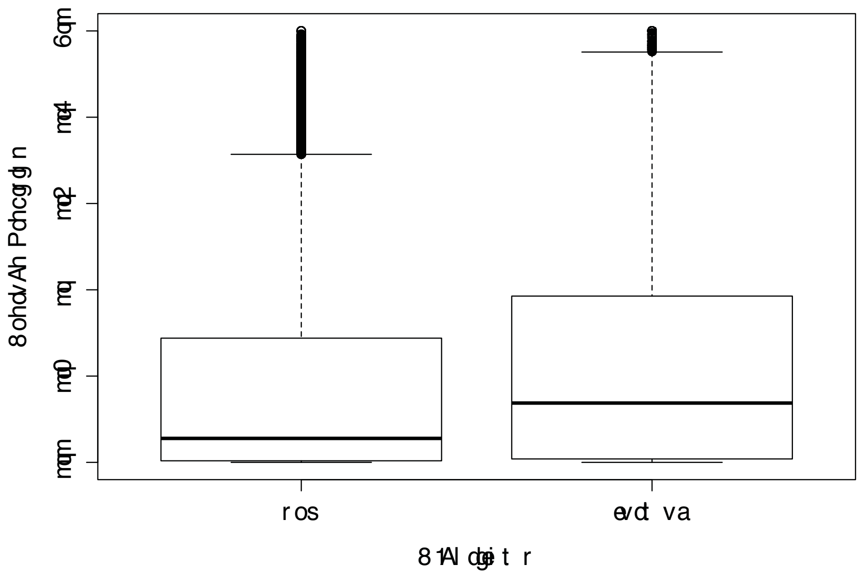


Fig. A.6: Distribution of average precision on unseen queries for each algorithm

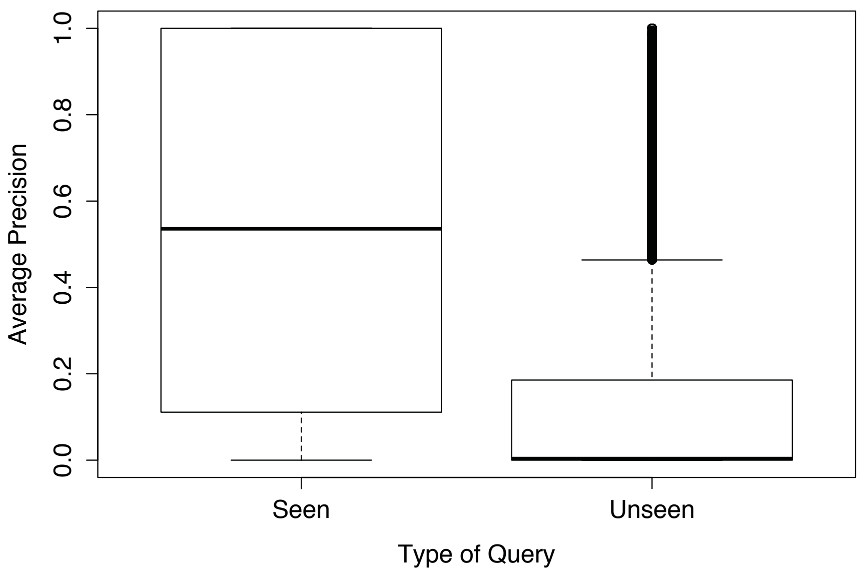


Fig. A.7: Distribution of average precision on seen and unseen queries

Table A.4: The mean average precision of each algorithm on each software system

	Rose	svd	Tarmaq	Friedman p-value
cisco	0.2063	0.1387	**0.3864**	< 0.00001
git	0.1042	0.1183	**0.2412**	< 0.00001
httpd	0.1035	0.2194	**0.3049**	< 0.00001
km	0.1711	0.3774	**0.3842**	< 0.00001
linux	0.1300	**0.5496**	0.3367	< 0.00001
mysql	0.1256	0.2062	**0.2958**	< 0.00001

7.6 Performance of Algorithms on Individual Systems (RQ3.5)

In this final results section we look closer at how performance varies over the individual software systems. To this end we present the *mean average precision* for each software system and algorithm combination in Table A.4. The last column in Table A.4 lists the p-value for a Friedman test of equal distributions between the algorithms. In all cases the p-value was found significant. To determine which algorithm did the best for each software system, we performed post-hoc Wilcoxon tests on each algorithm pair. The best performing algorithm is shown in bold. For all software systems, with the exception of Linux, Tarmaq is the top performer.

8 Discussion

In this section we will discuss some implications of the results presented in section 7.

8.1 Overall Performance on Average Precision (RQ3.1)

To answer RQ3.1 we measured the overall performance of Rose, Tarmaq and svd over all software systems, regardless of the type of query (seen vs unseen). We argue that this is the most realistic comparison, as it is hard to know the query type a priori. We therefore claim that a good evolutionary coupling based algorithm for change impact analysis should be agnostic to the type of query. The same argument also holds in terms of queries on different software systems. While there might be systems where evolutionary couplings are harder to analyze because of developer practices, the general algorithm should not be too sensitive to specific software systems.

8.2 Overall Performance on Time (RQ3.1)

When implementing a change recommendation engine in an industrial process, the time needed to generate recommendations is an important factor to consider when it comes to industry adoption, as a slow tool can deter users from frequent use.

In subsection 7.2 we found that Rose and Tarmaq executed in a fraction of a second on average, while the svd algorithm had an average execution time of 17 seconds. We observe that svd's execution time is unsatisfactory in a change-recommendation context, although it is certainly acceptable in other applications, such as for test-selection.

We also argue that the difference in execution time between Rose and Tarmaq is significant only statistically, not practically, and to this end both can be considered as providing real-time feedback.

8.3 Performance on Seen and Unseen Queries (RQ3.2-4)

In addition to investigating the overall performance, we also considered performance on seen and unseen queries separately. The discussion in subsection 7.3 and subsection 7.4 states the best-performing algorithm based on average precision, but does not give sufficient insights on the performance of each algorithm in practice. While the average precision is a good evaluator of the complete recommendation list produced by the algorithms, for this purpose, we introduce a quantification of practical use, which is easier to interpret than average precision. We define *top10-truepositive* as the number of times an algorithm correctly predicts at least one file in the top-10 of its recommendation list. The top10-truepositive metric, given in percentage, is shown in Table A.5.

We see that in over 90% of cases, at least one correctly predicted file will be in the top 10 for Rose and Tarmaq on seen queries. For svd the same is true only in about 50% of cases.

For all algorithms, performance measured by top10-truepositive degrades on unseen queries, which should be expected; however, given the average precision degradation we also saw in subsection 7.4. Tarmaq and svd are closer in performance here, but for about every 6th query, Tarmaq will predict at least one correct file in the top 10 that svd did not catch.

8.4 Performance of Algorithms on Individual Systems (RQ3.5)

In our final evaluation in subsection 7.6 we looked at each algorithm's performance on the individual software systems. Here we found that, with the exception of Linux, Tarmaq performed best on all systems. Finally, we remark the diversity expressed by the different system in terms of number of

Table A.5: Top10-truepositive: The percentage of times that each algorithm predicts at least one correct file in the top 10 on either seen or unseen queries

	seen	unseen
Rose	92%	0%
svd	56%	50%
Tarmaq	92%	65%

unique files, size of the average transaction, time between first and last transaction in the history, and the number of different languages used (Table A.1), and we argue that Tarmaq's performance on such diverse systems is satisfactory.

However, in the future we plan to achieve a better understanding of how individual software system characteristics might affect Tarmaq, especially in the case of Linux where svd achieves higher average precision than Tarmaq. A possible explanation for this result is the relatively short time frame of the history used with Linux, where the older commits are only a month old. Future work will consider if this short time frame could cause Tarmaq to assign low confidence and support to correct association rules.

8.5 Threats to Validity

We identified a set of threats that could affect the construct, internal, and external validity of our experimental results.

Threats to Construct Validity

One main threat could negatively affect the extent to which our experimental design measures the effectiveness of Tarmaq for the purpose of generating change recommendations.

Using Evolutionary Coupling for Software Change Impact Analysis: The main underlying assumption behind our experimental design is that evolutionary coupling infers meaningful dependencies from the transaction histories, which can in turn be used to generate effective change recommendations. While this has not been proven on a universal basis, research in the field showed that evolutionary coupling is an effective strategy for software change impact analysis (section 9).

Threats to Internal Validity

One main threat could negatively affect the conclusions on the cause-effect relationships derived from the experimental results.

Algorithms implementation: We compared the effectiveness of Tarmaq to that of the most commonly used alternatives for deriving change recommendations based transaction history, namely Rose and svd. However, we could not find publicly available implementations of these algorithms, and we re-implemented them based on the specification in the papers where such algorithms were introduced [7, 9]. In particular, we implemented the calculation of the decomposition matrices of svd using standard linear algebra libraries, which ensure number overflows are properly avoided. Repeated executions on abstract examples show that our implementations of Tarmaq, Rose and svd are correct.

Threats to External Validity

Four main threats could negatively affect the generalizability of the conclusions drawn.
Variation in software systems: We validated Tarmaq in two industrial systems from our user partners, and four large open source systems (subsection 6.1). These systems considerably vary in size and frequency of transactions, and have been selected in order to investigate the effectiveness of Tarmaq in a variety of software systems. However, even though our selected software systems display good variation, we very likely have not captured all variations.
Query Generation Process: When generating queries from the systems histories, we removed transactions larger than 30 files that could contain dependent files (subsection 6.3). However, as mentioned earlier, similar experimentation in the literature does not consider these large transactions, because on average they are likely to contain for the largest part unrelated files which would introduce noise when inferring dependencies [9].
Incomplete data from our industrial partners: We were not able to include the full history of transactions in the Kongsberg Maritime system. This is because such transactions were parsed from semi-structured free-text fields, which in a small number of cases contained incomplete data. However, the transactions we excluded for this reason constitute less than 0.05% of the total history of the KM system.
Length of history: We evaluated all algorithms using the last 10000 commits from each software system, rather than the entire available history. While this ensures consistency over the different systems, the included length might have also affected our evaluation of seen and unseen queries. We had to limit the number of commits for two reasons. (1) This was the number of commits provided to us by our industry partners, and we wanted to be consistent to this number also for the open-source systems. (2) For the svd algorithm, the overhead for generating singular value decompositions of large co-change matrices proved to be very high, and we saw the need to limit the number of

commits to keep experiment execution times to a manageable level.

9 Related Work

We distinguish related work on association rule mining, change impact analysis, evolutionary coupling, and mixed approaches.

Association Rule Mining: Since Agrawal et al's seminal paper introducing the concept [10], many techniques have been proposed, generally aimed at improving execution and memory efficiency. The most widely known include Apriori [17], which uses an efficient pre-computation of rule generation candidates, Eclat [18], which partitions the search space into smaller independent subspaces that can more efficiently be analyzed, and FPGrowth [19], which uses a compact tree structure (the FPtree) to encode the database and enable frequent patterns mining without candidate generation. All these apply frequent pattern mining on the complete dataset. A refinement is brought by so called *targeted association rule mining* techniques, which focus the generation of rules on a particular *query* supplied by the user [12, 20, 21]. These techniques filter transactions that are not related to the query from the database used for rule generation, enabling a drastic reduction of execution time [12]. Furthermore, these approaches are very suitable for evolving data like software change histories, because association rules are generated on a per-query basis, and are always "up-to-date" with the latest repository status. A more detailed discussion of advances in pattern mining is outside the scope of this paper. For more details, we refer to a recent survey by Silva et al. [22].

Change Impact Analysis: Zimmerman et al. introduce Rose [9], the work most closely related to ours. Rose applies targeted association rule mining to the problem of deriving developer change recommendations for a user specified query. It uses constraints to filter out transactions that do not contain any of the files of the query. The same constraint is used to generate only rules whose antecedent contains *all* the files in the query. As discussed in section 4, the downside of this approach is that Rose generates *no* rules if no transaction in the history is a superset of the query.

Ying et al. [13] describe a technique that mines frequent patterns in the change history of a system to recommend potentially relevant source code to a developer that is performing a software maintenance task. Their algorithm uses a *FPtree* structure to efficiently represent the set of files frequent in the history [19]. Similar to Rose, this algorithm generates no rules if no transaction in the history is a superset of the query, i.e. it suffers from the limitation discussed in section 4.

Sherriff et al. [7] present an approach to change recommendation that identifies couplings of related files using a svd of the co-change matrix. This matrix encodes the number of times any two files changed in the same transaction. This algorithm does not suffer from the limited applicability discussed in section 4, but it is rather computationally expensive, as demonstrated by the results of our empirical evaluation (section 7). Moreover, the SVD has to be recomputed after an update of the change history.

Evolutionary Coupling: There is a body of work on identifying *evolutionary coupling* (also referred to as *logical coupling*). All these have in common that they are based on some measure of co-change. Example measures include course-, and fine-grained co-change information [11, 23, 24], code-churn [25], and interaction with an IDE [4].

Gall et al. used release information to detect logical coupling between 20 releases of a large Telecommunication Switching System [11]. They later continued this analysis to discover architectural weaknesses in source code (e.g., amount of modularization) [25]. The coupling were primarily found through analyzing sequences of releases in which modules were changed together. Furthermore, couplings were also identified on a class level through analyzing when and who (author/date) that made class changes.

Hassan and Holt present several heuristics for predicting ripple effects that result from source code changes [26]. In addition to evolutionary coupling (described as "historical co-changes" in the paper), three other heuristics were also investigated, whereas one used static dependencies such as *Call/Use/Define* relations, and another used code-layout to identify couplings. Of all 4 heuristics, the use of evolutionary couplings gave the highest recall score, meaning it correctly identified the most couplings (avg. 87%).

Jafar et al. [27] perform an exploratory study on co-changes at file level granularity. They introduce two timing related patterns for co-changes that can help to more accurately mine transactions in a change history.

Mixed Approaches: Hipikat [28] integrates various developer related artifacts, such as change history, email discussions and issue tracking systems. Vector-based information retrieval techniques are used to mine relations between artifacts. Additional relations are created using heuristics, such as the matching of issue IDs in commit messages to issue reports in bugzilla. Hipikat uses this cross-indexed *project memory* to recommend relevant artifacts for a task , either directly from a query, or automatically based on a developer's working context (e.g., documents open in an IDE).

Kagdi et al. [29] combine history based evolutionary coupling with so called *conceptual coupling* which is derived using information retrieval techniques on a single version (i.e, a release) of a software system. They show that the combination of these two techniques provides statistically significant improvements in accuracy over the individual techniques. Mondal et al. [30] combine association rule mining with *change correspondence*, a measure for the extent to which identifiers and constants in co-changed entities overlap. This is a lightweight form of conceptual coupling, which is then used to prioritize association rules, as a more source-code aware version of the standard support and confidence measures.

Although originally developed for other contexts, there is no reason why these orthogonal measures and additional sources could not be used in combination with the technique we propose in this paper, and achieve similar benefits as in their original application.

10 Concluding Remarks

With new techniques and data-sources, progress is being made on improving Change Impact Analysis (CIA). The use of evolutionary coupling as a driver of CIA is a promising direction that can address some caveats of traditional static/dynamic dependency analysis. In particular, the use of evolutionary coupling is inherently language agnostic, and in general can potentially find couplings where static/dynamic approaches cannot find a coupling because of a lack of explicit data/control flow. This is a considerable advantage given the increasing heterogeneity of today's software systems.

In this paper we present an algorithm (Tarmaq) for CIA using a *generalized analysis* of evolutionary coupling with respect to some change-scenario (changed files). The use of Tarmaq for CIA promises a best effort analysis of the evolutionary coupling given any change-scenario. The contributions of this paper are the following: (1) We provided a classification of two different change-scenarios for change impact analysis. (2) We provided an empirical evaluation of the frequency of these change-scenarios. (3) We introduced an algorithm (Tarmaq) that can analyze the evolutionary coupling of any change-scenario, and therefore can support CIA for most change-scenarios. (4) We provided a comprehensive evaluation of Tarmaq on two industrial software systems from our industry partner and four open source systems.

Directions for Future Research: In future work we would like to address the following: (1) We plan to conduct a larger empirical evaluation of Tarmaq, both in the number of software systems and in the number of evaluated factors. We would, for example, like to explore the effect of history-size and query-size on performance, and attempt to classify software systems in terms of how applicable our approach might be. (2) We also plan to explore methods for further increasing overall performance, and especially performance on unseen queries. (3) Third, we hope to directly compare CIA based on evolutionary coupling with static/dynamic dependency analysis. (4) Next we plan to apply Tarmaq on other problems, for example test-selection. (5) Finally, future work will explore existing alternative interestingness measures that might replace support/confidence, and see how they perform.

Acknowledgement: This work is supported by the Research Council of Norway through the EvolveIT project (#221751/F20) and the Certus SFI. Dr. Binkley is supported by NSF grant IIA-1360707 and a J. William Fulbright

award.

References

[1] G. Canfora and L. Cerulo, "Impact Analysis by Mining Software and Change Request Repositories," in *International Software Metrics Symposium (METRICS)*. IEEE, 2005, pp. 29–37. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1509307http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1509307

[2] M.-A. Jashki, R. Zafarani, and E. Bagheri, "Towards a more efficient static software change impact analysis method," in *ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE)*. ACM, 2008, pp. 84–90. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1512475.1512493

[3] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, "Chianti: a tool for change impact analysis of java programs," in *ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2004, pp. 432–448. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1035292.1029012

[4] M. B. Zanjani, G. Swartzendruber, and H. Kagdi, "Impact analysis of change requests on source code based on interaction and commit histories," in *International Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 162–171. [Online]. Available: http://dl.acm.org/citation.cfm?id=2597096http://dx.doi.org/10.1145/2597073.2597096

[5] D. W. Binkley, N. Gold, M. Harman, S. Islam, J. Krinke, and S. Yoo, "ORBS and the Limits of Static Slicing," in *International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2015, pp. 1–10. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2635868.2635893

[6] A. R. Yazdanshenas and L. Moonen, "Crossing the boundaries while analyzing heterogeneous component-based software systems," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011, pp. 193–202. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2011.6080786http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6080786

[7] M. Sherriff and L. Williams, "Empirical Software Change Impact Analysis using Singular Value Decomposition," in *International Conference on Software Testing, Verification and Validation (ICST)*. IEEE,

2008, pp. 268–277. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4539554

[8] R. Srikant and R. Agrawal, "Mining generalized association rules," in *International Conference on Very Large Data Bases (VLDB)*, 1995, pp. 407–419. [Online]. Available: http://wwwqbic.almaden.ibm.com/cs/projects/iis/hdb/Publications/papers/vldb95{_}tax{_}rj.pdf

[9] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1463228

[10] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD International Conference on Management of Data*. ACM, 1993, pp. 207–216. [Online]. Available: http://portal.acm.org/citation.cfm?doid=170035.170072

[11] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 1998, pp. 190–198. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=738508

[12] R. Srikant, Q. Vu, and R. Agrawal, "Mining Association Rules with Item Constraints," in *International Conference on Knowledge Discovery and Data Mining (KDD)*. AASI, 1997, pp. 67–73.

[13] A. T. T. Ying, G. Murphy, R. T. Ng, and M. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1324645

[14] T. Zimmermann, "Preprocessing CVS data for fine-grained analysis," in *International Workshop on Mining Software Repositories (MSR)*, vol. 2004. IEE, 2004, pp. 2–6. [Online]. Available: http://link.aip.org/link/IEESEM/v2004/i917/p2/s1{&}Agg=doihttp://digital-library.theiet.org/content/conferences/10.1049/ic{_}20040466

[15] L. A. Granka, T. Joachims, and G. Gay, "Eye-tracking analysis of user behavior in WWW search," in *International conference on Research and development in information retrieval (SIGIR)*. ACM, 2004, p. 478. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1008992.1009079

[16] J. O. Woods and the Ruby Science Foundation, "NMatrix: A dense and sparse linear algebra library for the Ruby programming language," 2013. [Online]. Available: http://sciruby.com

[17] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” in *International Conference on Very Large Data Bases (VLDB)*, 1994, pp. 487–499.

[18] M. J. Zaki, “Scalable algorithms for association mining,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 3, pp. 372–390, 2000. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=846291

[19] J. Han, J. Pei, Y. Yin, and R. Mao, “Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach,” *Data Mining and Knowledge Discovery*, vol. 8, no. 1, pp. 53–87, 2004. [Online]. Available: http://link.springer.com/10.1023/B:DAMI.0000005258.31418.83

[20] A. Hafez, J. Deogun, and V. V. Raghavan, “The Item-Set Tree: A Data Structure for Data Mining,” in *Data Warehousing and Knowledge Discovery*, ser. LNCS. Springer, 1999, vol. 1676, pp. 183–192. [Online]. Available: http://link.springer.com/10.1007/3-540-48298-9

[21] M. Kubat, A. Hafez, V. V. Raghavan, and J. Lekkala, “Itemset trees for targeted association querying,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1522–1534, nov 2003. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1245290

[22] A. Silva and C. Antunes, “Constrained pattern mining in the new era,” *Knowledge and Information Systems*, vol. 47, no. 3, pp. 489–516, 2016. [Online]. Available: http://link.springer.com/10.1007/s10115-015-0860-5

[23] D. Beyer and A. Noack, “Clustering Software Artifacts Based on Frequent Common Changes,” in *International Workshop on Program Comprehension (IWPC)*. IEEE, 2005, pp. 259–268. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1421041

[24] R. Robbes, D. Pollet, and M. Lanza, “Logical Coupling Based on Fine-Grained Change Information,” in *Working Conference on Reverse Engineering (WCRE)*. IEEE, 2008, pp. 42–46. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4656392

[25] H. Gall, M. Jazayeri, and J. Krajewski, “CVS release history data for detecting logical couplings,” in *International Workshop on Principles of Software Evolution (IWPSE)*. IEEE, 2003, pp. 13–23. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1231205

[26] A. E. Hassan and R. Holt, "Predicting change propagation in software systems," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2004, pp. 284–293. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1357812

[27] F. Jaafar, Y.-G. Guéhéneuc, S. Hamel, and G. Antoniol, "An Exploratory Study of Macro Co-changes," in *Working Conference on Reverse Engineering (WCRE)*. IEEE, oct 2011, pp. 325–334. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6079858

[28] D. Cubranic, G. Murphy, J. Singer, and K. Booth, "Hipikat: a project memory for software development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 446–465, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1463229

[29] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. L. Collard, "Blending conceptual and evolutionary couplings to support change impact analysis in source code," in *Working Conference on Reverse Engineering (WCRE)*, 2010, pp. 119–128.

[30] M. Mondal, C. K. Roy, and K. A. Schneider, "Improving the detection accuracy of evolutionary coupling by measuring change correspondence," in *Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, 2014, pp. 358–362. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6613853http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6747194

Paper B

Improving Change Recommendation using Aggregated Association Rules

Thomas Rolfsnes, Leon Moonen, Stefano Di Alesio,
Razieh Behjati and Dave W. Binkley

Accepted for publication in the main research track of the 13th International Conference on Mining Software Repositories (MSR), May 14–15, 2016.
Further invited and submitted to the MSR 2016 special issue published by the Springer journal Empirical Software Engineering (EMSE).

© 2016 IEEE
The layout has been revised.

Abstract

As the complexity of software systems grows, it becomes increasingly difficult for developers to be aware of all the dependencies that exist between artifacts (e.g., files or methods) of a system. Change recommendation has been proposed as a technique to overcome this problem, as it suggests to a developer relevant source-code artifacts related to her changes. Association rule mining has shown promise to deriving such recommendations by uncovering relevant patterns in the system's change history. The strength of the mined association rules is captured using a variety of interestingness measures. However, state-of-the-art recommendation engines typically use only the rule with the highest interestingness value when more than one rule applies. In contrast, we argue that when multiple rules apply, this indicates collective evidence, and aggregating those rules (and their evidence) will lead to more accurate change recommendation.

To investigate this hypothesis we conduct a large empirical study of 15 open source software systems and two systems from our industry partners. We evaluate association rule aggregation using four variants of the change history for each system studied, enabling us to compare two different levels of granularity in two different scenarios. Furthermore, we study 40 interestingness measures using the rules produced by two different mining algorithms. The results show that (1) between 13% and 90% of change recommendations can be improved by rule aggregation, (2) rule aggregation almost always improves change recommendation for both algorithms and all measures, and (3) fine-grained histories benefit more from rule aggregation.

1 Introduction

The evolution of a software system is accompanied by continuous growth in the number and complexity of interactions between system artifacts. Thus, over time, it becomes increasingly challenging for developers to manage the impact of changes made to the system. To address this problem, Change Impact Analysis (CIA) has been proposed to identify software artifacts (e.g., files, classes, and methods) affected by a given change [1–4]. CIA is typically used to derive a *software change recommendation* as direct feedback to a developer regarding artifacts that may also be in need of change.

Classical CIA employs static and dynamic dependence analysis [5]. For example, by identifying the methods that call a changed method. However, both static and dynamic dependence analysis are generally language specific, making them unsuitable for heterogeneous software systems [6]. In addition, because of its conservative nature static analysis is known to over-approximate solutions [7], while dynamic analysis can involve considerable run-time overhead [8].

These limitations have increased interest in alternative approaches [9].

Prominent among these is the identification of dependences using *evolutionary coupling*. Such couplings are based on *how* a software system changes over time, something that is missed by static and dynamic dependence analysis. In essence, evolutionary coupling taps into the developers' inherent knowledge of the dependencies in the system. This *co-change* knowledge can manifest itself in several ways: commits, bug-reports, context-switches in an IDE, etc. It can thus be extracted, for example, from the project's version control system [10], its issue tracking database, or by instrumenting the development environment [11].

This paper explores the use of co-change information extracted from *git* repositories as the basis for uncovering the evolutionary coupling. Doing so exploits the fact that dependent artifacts are likely to change together. Specifically, we mine evolutionary couplings using *association rule mining* [12], an unsupervised machine learning approach that reveals relations among items in a data set. The relative importance of mined rules is typically captured by an *interestingness measure* (e.g., *support* and *confidence*), which seeks to capture the relative utility of each mined rule [13–15]. In scenarios where multiple rules can be applied, it is common to consider only the single rule with the highest interestingness value [16]. In contrast, we hypothesize that the aggregation of such rules can be exploited to provide improved recommendations.

Contributions: This article builds upon our previous work with *association rule aggregation* [17]. In particular, it extends our previous work in six key respects: (1) we significantly scale up the empirical study of association rule aggregation in the context of change recommendation (2) we include a study of aggregation performance on the level of individual software systems (3) we introduce and study a new aggregation function, the *Hyper Cumulative Gain* (4) we study the effect that history granularity (files, methods etc.) has on the precision gained from rule aggregation (5) we formally prove that all studied aggregation functions satisfy our previously proposed aggregation properties, and finally (6) we extend our review of the related work.

Overview: The rest of this article is organized as follows: section 2 provides background on targeted association rule mining. section 3 describes limitations of the current state-of-art approaches to association rule mining. section 4 overviews the interestingness measures used to weigh the mined association rules. section 5 introduces the aggregation of association rules into hyper-rules, while section 6 presents the aggregation functions considered in the experiments. section 7 describes the setup of our empirical evaluation whose results are presented and discussed in section 8. section 9 discusses potential threats that could affect the validity of our conclusions, and section 10 presents related work. Finally, section 11 concludes the article with final remarks and outlines future work.

2 Association Rule Mining

Agrawal et al. introduced the concept of *association rule mining* as the discipline aimed at inferring relations between *entities* of a data set [12]. *Association rules* are implications of the form $A \rightarrow B$, where A is referred to as the *antecedent*, B as the *consequent*, and A and B are disjoint sets of entities. For example, consider the classic application of analyzing shopping cart data; if multiple transactions include bread and butter then a potential association rule is *bread* $\rightarrow$ *butter*. This rule can be read as *"if you buy bread, then you are also likely to buy butter."*

In the context of mining evolutionary coupling from historical co-change data, the entities are the files of the system[1] and the sequence (history) $\mathcal{T}$ of transactions, is the sequence of past *commits*. More specifically, a transaction $T \in \mathcal{T}$ is the set of files that were either changed or added while addressing a given bug or feature addition, hence creating a *logical dependence* between the files [18].

As originally defined [12], association rule mining generates rules that express patterns in a complete data set. However, some applications can exploit a more focused set of rules. *Targeted association rule mining* [19] focuses the generation of rules to those that satisfy a given constraint, e.g., stating that the antecedent of all mined rules has to belong to a particular set of files. Doing so reduces the number of rules generated and thus can significantly improve the rule generation time [19].

When generating change recommendations, rule constraints are based on a *change set*: the set of modified files since the last commit. In this case, only rules with at least one changed artifact in the antecedent are generated. The output of a change recommendation is the set of files that are historically changed along with the elements of the change set. For example, given the change set $\{a, b, c\}$, a change recommendation would consist of the files that were changed when a, b, and c were changed. The recommended files are those found in the consequent of the mined rules, and these files are typically ranked based on the rule's *interestingness value*.

To the best of our knowledge, only a few targeted association rule mining algorithms have been considered in the context of change recommendation: Zimmerman et al. [20], Ying et al. [21], and Rolfsnes et al. [22] (our previous work). In contrast, simple *co-change* algorithms have been applied in a variety of domains [18, 23–25]. The existing targeted association rule mining algorithms and the simple co-change algorithms differ in terms of the subsets of

[1] Other levels of granularity are possible as our algorithms are granularity agnostic. Thus, our initial description at the file level is without loss of generality. Provided suitably co-change data the algorithms can relate methods or variables just as well as files, a fact which will be exploited later on in the paper.

the change set used to form the rule antecedents. Consider, for example, the subsets of the change-set $C = \{a, b, c, d\}$:

$$\begin{aligned} powerset(C) = \{ & \{\}, && \text{(B.1)} \\ & \{a\}, \{b\}, \{c\}, \{d\}, && \text{(B.2)} \\ & \{a,b\}, \{a,c\}, \{a,d\}, \{b,c\}, \{b,d\}, \{c,d\}, && \text{(B.3)} \\ & \{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}, && \text{(B.4)} \\ & \{a,b,c,d\}\} && \text{(B.5)} \end{aligned}$$

Of C's subsets, both Zimmerman's and Ying's algorithms only consider rules based on line B.5 (i.e., rules of the form $\{a, b, c, d\} \rightarrow X$) because these techniques constrain the antecedent to be equal to the change set. At the other end of the spectrum, co-change algorithms consider rules from the singleton sets in line B.2, such as $\{a\} \rightarrow X$ or $\{b\} \rightarrow X$. (As a notational convenience, singleton sets are often written without brackets as in "$a \rightarrow X$".) In previous work, we introduced Tarmaq, the most versatile among these algorithms [22]. Tarmaq uses the set from any one of lines B.2, B.3, B.4, or B.5. The particular line used is dynamically chosen based on the maximal overlap with the change set [22].

In this paper we study only Co-Change and Tarmaq. Zimmerman's Rose algorithm is not included because its behavior is subsumed by Tarmaq (i.e., whenever Rose is able to make a recommendation, Tarmaq makes the same recommendation, but Tarmaq is able to generate recommendations when Rose is not [22]). To provide some intuition on the behavior of Co-Change and Tarmaq we provide the following example:

Example 1 (Co-Change and Tarmaq). *Given a query* Q *of changed artifacts,* Co-Change *and* Tarmaq *work as follows:*

Co-Change*: for each artifact $q \in Q$, find the set of artifacts* C *not in Q that have changed with* q *in the past, then for each $c \in C$ create rules such that the left hand side is equal to* q*, and the right hand side is equal to* c.

Tarmaq*: find the transactions with the largest intersection with* Q*, then create rules such that the left hand size is equal to the intersection, and the right hand side is equal to the respective difference ("leftover" artifacts of each transaction).*

For example, assume that artifacts a, b, and f have been changed by a developer, yielding the query $Q = \{a, b, f\}$, then given the following change history, Co-

Change *and* Tarmaq *will output the following rules:*

Change History		Co-Change	Tarmaq
TXID	*Artifacts*	$a \rightarrow Y$	$a, b \rightarrow X$
TX_1	$\{a, Y\}$	$a \rightarrow X$	$b, f \rightarrow X$
TX_2	$\{a, b, X\}$	$b \rightarrow X$	
TX_3	$\{b, f, X\}$	$f \rightarrow X$	

For Co-Change*, a has changed with both Y and X, resulting in the two first rules, while b and f has changed with X, resulting in the two last rules. For* Tarmaq*, the largest intersections (of size two) can be found in TX_2, and TX_3, and the difference with each respective intersection and transaction is X, resulting in the two rules.*

3 Problem Description

In complex software systems, as well as systems with many active developers, it can be challenging for each individual developer to be aware of all dependencies that exist in the software. To aid a developer, a *change recommendation* can be made based on recent changes. The application of association rule mining to change recommendation involves looking for the *evolutionary coupling* between artifacts (files, methods, etc.) of a system. This search considers artifacts coupled if and only if they have changed together in the past. Furthermore, the *strength* of a coupling is given by an *interestingness value*. For example, how frequently the rule's artifacts change together.

In our previous work [22], which sought to find evolutionary couplings through association rule mining of a system's version history, we noticed that there are often rules with different *antecedents*, but the same *consequent*. For example, consider the following rules involving artifacts a, b, and c:

$$r_1 = \{a\} \rightarrow \{c\}$$
$$r_2 = \{b\} \rightarrow \{c\}$$

which can be interpreted as "if you change a, consider changing c," and "if you change b consider changing c." Given the change set $\{a, b\}$, existing recommendations systems will select *one* of the two rules, that recommend c. However, we conjecture that doing so may be a mistake. For example, having multiple applicable rules for the same consequent potentially provides increased evidence that the consequent is relevant. We hypothesize that this increased evidence can be captured by the *aggregation* of rules into *hyper-rules*, whose use will lead to more accurate recommendations. In terms of the example, we seek to combine rules r_1 and r_2 into the *hyper-rule* r_3 that captures the cumulative evidence that *c* should be recommended for change when *a* and *b* are changed.

A concrete example will help illustrate our goal and also provide a better intuition into the value of association rule aggregation. The example involves a sequence of past transactions that each include a set of artifacts that changed together. The example also motivates the need to aggregate the interestingness values (defined in section 4) of the rules to produce an interestingness value for the resulting hyper-rule. The example does this using, as a simple interestingness value, the percentage of the transactions that give rise to the rule.

Example 2. *Consider the following (historic) sequence of transactions:*

$$\mathcal{T} = [\{a, x\}, \{b, y\}, \{c, y\}, \{d, y\}, \{a, x\}]$$

and the change set $C = \{a, b, c, d\}$ *where, based on* $\mathcal{T}$ *and* C*, the following rules have been mined (the interestingness of each is given in parentheses):*

$$\begin{aligned} a &\to x \quad (40\%) \\ b &\to y \quad (20\%) \\ c &\to y \quad (20\%) \\ d &\to y \quad (20\%) \end{aligned}$$

In these rules all the artifacts that occur in an antecedent are part of change set C *while all artifacts that occur in a consequent are potentially impacted by the change with a certainty reflected by the rule's interestingness value.*

Clearly, without aggregation, *x* is recommended above *y*, because it has changed two times with an item in the change set (*a*), while *y* had changed at most once with *any individual* item of the change set. However, *y* has changed more times with *at least one* item of the change set. Therefore, there is combined evidence that *y* should be recommended above *x*.

Generalizing this example, our goal is to aggregate mined association rules into hyper-rules, which combine evidence and ultimately provide more accurate recommendations. To this end, the remainder of this paper investigates the impact of three aggregation techniques on the performance of two association rule mining algorithms using a collection of 40 interestingness-measures.

4 Interestingness Measures

The relative value of the rules mined via targeted association rule mining is given by an *interestingness measure*. In Agrawal et al.'s seminal paper on association rule mining [12], two interestingness measures were introduced, *support* and *confidence*.

Definition 1 (Support). *Given a sequence of transactions $\mathcal{T}$, the* support *of the rule $A \rightarrow B$ is defined as the number of transactions where the union of the antecedent and consequent is a subset, divided by the total number of transactions. Therefore, support represents the probability of $A \cup B$ being a subset in a transaction:*

$$support(A \rightarrow B) \stackrel{def}{=} \frac{|\{T \in \mathcal{T} : \{A \cup B\} \subseteq T\}|}{|\mathcal{T}|}$$

Intuitively, the higher the support, the more likely the rule is to hold, while rules with low support identify weaker relations. For this reason, a minimum threshold on support is often used to filter out uninteresting rules.

Definition 2 (Confidence). *Given a sequence of transactions $\mathcal{T}$, the* confidence *of the rule $A \rightarrow B$ is defined as the number of transactions with the union of A and B as a subset, divided by the number of transactions where A is a subset. Therefore, the confidence represents the conditional probability of B being a subset in a transaction, given that A is a subset of that transaction:*

$$confidence(A \rightarrow B) \stackrel{def}{=} \frac{|\{T \in \mathcal{T} : \{A \cup B\} \subseteq T\}|}{|\{T \in \mathcal{T} : A \subseteq T\}|}$$

Since the introduction of targeted association rule mining, a large number of alternative interestingness measures have been proposed. However, all these measures can be defined using the same set of basic probabilistic quantities. Indeed, the interestingness measures of a rule $A \rightarrow B$ build upon the following probabilities:

$P(A)$: the likelihood of A changing in the history.

$P(B)$: the likelihood of B changing in the history.

$P(A, B)$: the likelihood of A and B changing together in the history.

As shown in Table B.1, the other probabilities used in the measure definitions can be inferred from these three. For example, *support* is the probability $P(A, B)$, which is the probability that a transaction includes both A and B. Likewise, *confidence* is the probability $P(B|A)$, which is the conditional probability that B is in a transaction given that A is in the same transaction. Several measures also account for the non-occurrence of the antecedent or consequent. For example, the *causal support* is defined as $P(A, B) + P(\neg A, \neg B)$.

Table B.1: Overview of probabilistic building blocks used to define the interestingness measures of Table B.2

Probability	Definition
$P(A)$	$\frac{\lvert\{T\in\mathcal{T}:A\subseteq T\}\rvert}{\lvert\mathcal{T}\rvert}$
$P(B)$	$\frac{\lvert\{T\in\mathcal{T}:B\subseteq T\}\rvert}{\lvert\mathcal{T}\rvert}$
$P(A,B)$	$\frac{\lvert\{T\in\mathcal{T}:\{A\cup B\}\subseteq T\}\rvert}{\lvert\mathcal{T}\rvert}$
$P(\neg A)$	$1-P(A)$
$P(\neg B)$	$1-P(B)$
$P(\neg A,\neg B)$	$1-P(A)-P(B)+P(A,B)$
$P(\neg A,B)$	$P(B)-P(A,B)$
$P(A,\neg B)$	$P(A)-P(A,B)$

Conditional Probabilities	Definition
$P(A\|B)$	$\frac{P(A,B)}{P(B)}$
$P(B\|A)$	$\frac{P(B,A)}{P(A)}$
$P(\neg A\|B)$	$\frac{P(\neg A,B)}{P(B)}$
$P(\neg B\|A)$	$\frac{P(\neg B,A)}{P(A)}$
$P(A\|\neg B)$	$\frac{P(A,\neg B)}{P(\neg B)}$
$P(B\|\neg A)$	$\frac{P(B,\neg A)}{P(\neg A)}$
$P(\neg A\|\neg B)$	$\frac{P(\neg A,\neg B)}{P(\neg B)}$
$P(\neg B\|\neg A)$	$\frac{P(\neg B,\neg A)}{P(\neg A)}$

Table B.2: Overview of the 40 interestingness measures considered in our study (continued on next page). The notation $[min..mid..max]$ is used to provide the range of each interestingness measure, min/max provides the minimum and maximum value respectively, mid indicates the point of no correlation. If only $[min..max]$ is used, the point of no correlation is given by min.

Interestingness Measure	Range	Definition
Added Value [14]	$[-0.5..0..1]$	$P(B\|A) - P(B)$
Causal Confidence [26]	$[0..1]$	$\frac{1}{2} * (P(B\|A) + P(\neg A\|\neg B))$
Causal Support [26]	$[0..1]$	$P(A,B) + P(\neg A, \neg B)$
Collective Strength [27]	$[0..1..\infty)$	$\frac{P(A,B)+P(\neg B\|\neg A)}{P(A)*P(B)+P(\neg A)*P(\neg B)} * \frac{1-P(A)*P(B)-P(\neg A)*P(\neg B)}{1-P(A,B)-P(\neg B\|\neg A)}$
Confidence [12]	$[0..1]$	$P(B\|A)$
Conviction [28]	$[0..\infty)$	$\frac{P(A)*P(\neg B)}{P(A,\neg B)}$
Cosine [14]	$[0..1]$	$\frac{P(A,B)}{\sqrt{P(A)*P(B)}}$
Coverage [15]	$[0..1]$	$P(A)$
Descriptive Confirmed Confidence [26]	$[-1..0..1]$	$P(B\|A) - P(\neg B\|A)$
Difference Of Confidence [29]	$[-1..0..1]$	$P(B\|A) - P(B\|\neg A)$
Example and Counterexample Rate [30]	$(-\infty..0..1]$	$\frac{(P(A,B)-P(A,\neg B))}{P(A,B)}$
Gini Index [31]	$[0..1]$	$P(A) * (P(B\|A)^2 + P(\neg B\|A)^2) + P(\neg A) * (P(B\|\neg A)^2 + P(\neg B\|\neg A)^2) - P(B)^2 - P(\neg B)^2$
Imbalance Ratio [32]	$[0..1]$	$\frac{\|P(A\|B)-P(B\|A)\|}{P(A\|B)+P(B\|A)-P(A\|B)*P(B\|A)}$
Interestingness Weighting Dependency		
(with parameters k=2, m=2) [33]	$[0..1]$	$(\frac{P(B\|A)}{P(B)})^{(k-1)} * (P(A,B))^m$
J Measure [34]	$[0..1]$	$P(A,B) * log(\frac{P(B\|A)}{P(B)}) + P(A,\neg B) * log(\frac{P(\neg B\|A}{P(\neg B})$
Jaccard [35]	$[0..1]$	$\frac{P(A,B)}{(P(A)+P(B)-P(A,B))}$
Kappa [36]	$[-1..0..1]$	$\frac{P(A,B)+P(\neg A,\neg B)-P(A)*P(B)-P(\neg A)*P(\neg B)}{1-P(A)*P(B)-P(\neg A)*P(\neg B)}$
Klösgen [37]	$[-1..0..1]$	$\sqrt{P(A,B)} * max(P(B\|A) - P(B), P(A\|B) - P(A))$

Table B.2: Overview of the 40 interestingness measures considered in our study (continued from prev. page).

Interestingness Measure	Range	Definition
Kulczynski [38]	[0..1]	$\frac{P(A,B)}{2} * (\frac{1}{P(A)} + \frac{1}{P(B)})$
Laplace Corrected Confidence [39]	[0..1]	$\frac{P(A,B)+1}{P(B)+2}$
Least Contradiction [40]	$(-\infty..0..1]$	$\frac{P(A,B)-P(A,\neg B)}{P(B)}$
Leverage [41]	$[-1..0..1]$	$P(B\|A) - P(A) * P(B)$
Lift [28]	$[0..1..\infty)$	$\frac{P(A,B)}{P(A)*P(B)}$
Linear Correlation Coefficient [42]	$[-1..0..1]$	$\frac{P(A,B)-P(A)*P(B)}{\sqrt{P(A)*P(B)*P(\neg A)*P(\neg B)}}$
Loevinger [43]	$[-1..0..1]$	$1 - \frac{P(A)*P(\neg B)}{P(A,\neg B)}$
Odd Multiplier [30]	$[0..\infty)$	$\frac{P(A,B)*P(\neg B)}{P(B)*P(A,\neg B)}$
Odds Ratio [44]	$[0..1..\infty)$	$\frac{P(A,B)*P(\neg A,\neg B)}{P(A,\neg B)*P(\neg A,B)}$
One Way Support [45]	$[-1..0..\infty)$	$P(B\|A) * log_2(\frac{P(A,B)}{P(A)*P(B)})$
Piatetsky-Shapiro [41]	$[-0.25..0..0.25]$	$P(A,B) - P(A) * P(B)$
Prevalence [15]	[0..1]	$P(B)$
Recall [15]	[0..1]	$P(A\|B)$
Relative Risk [15]	$[0..\infty)$	$\frac{P(B\|A)}{P(B\|\neg A)}$
Sebag Schoenauer [46]	$[0..\infty)$	$\frac{P(A,B)}{P(A,\neg B)}$
Specificity [15]	[0..1]	$P(\neg B\|\neg A)$
Support [12]	[0..1]	$P(A,B)$
Two Way Support [45]	$[-1..0..1]$	$P(A,B) * log_2(\frac{P(A,B)}{P(A)*P(B)})$
Varying Rates Liaison [47]	$[-1..0..\infty)$	$\frac{P(A,B)}{P(A)*P(B)} - 1$
Yules Q [48]	$[-1..0..1]$	$\frac{\text{odds ratio}-1}{\text{odds ratio}+1}$
Yules Y [49]	$[-1..0..1]$	$\frac{\sqrt{\text{odds ratio}}-1}{\sqrt{\text{odds ratio}}+1}$
Zhang [50]	$[-1..0..1]$	$\frac{P(A,B)-P(A)*P(B)}{max(P(AB)*P(\neg B),P(B)*P(A,\neg B))}$

A complete list of the interestingness measures used in our study and their definitions is given in Table B.2.

There is one final detail related to the interestingness measures that is relevant to our discussion: the *range* of a measure, and specifically its ability to measure either negative, positive, or no correlation between a rule's antecedent and consequent. Early measures such as support and confidence focus on positive correlations. However, it is also possible to consider negative correlations. These would concern rules that capture, for example, *"if a changes, then it is unlikely that you need to change b"*.

The range of most measures falls into one of a few categories. Most existing measures (e.g., *support*) range between 0 and 1. This [0..1] range is also the easiest to interpret as a correlation, where 0 naturally indicates no correlation and any higher value the degree of positive correlation. Another common range is [-1..0..1], where 0 again indicates no correlation, but negative correlation is also possible. In addition, there also exist ranges such as [0..1..∞), where 1 indicates no correlation and the maximum value is unbounded.

Piatetsky and Shapiro formalize this notion using four properties that they assert all interestingness measures should satisfy [41]. The four properties use V to denote the value produced by an interestingness measure:

No correlation: $V = 0$ when $P(A)$ and $P(B)$ are statistically independent (i.e., when $P(A, B) = P(A) \cdot P(B)$).

Positive correlation: When $P(A)$ and $P(B)$ remain unchanged, but $P(A, B)$ monotonically increase, V should also monotonically increase.

Negative correlation: When $P(B)$ and $P(A, B)$ remain unchanged, but $P(A)$ monotonically decrease, V should also monotonically decrease.

Negative correlation: When $P(A)$ and $P(A, B)$ remain unchanged, but $P(B)$ monotonically decrease, V should also monotonically decrease.

Existing interestingness measures satisfy these properties to a varying degrees, especially with respect to the way positive and negative correlation are captured [14]. For example, 23 of the 40 measures shown in Table B.2, capture only positive correlations. Furthermore, based on the empirical data we collected, only three of the remaining 17 measures actually produced negative correlations in practice.[2]

[2]The three measures are: *descriptive confirmed confidence*, *example and counterexample rate*, and *least contradictions*. Other able measures also sometimes produced negative values, although quite rarely.

5 Association Rule Aggregation

To assess the value of aggregating the evidence provided by a collection of conventional association rules, we introduce the concept of a *hyper-rule*, which provides an effective summary of a set of constituent rules. When forming hyper-rules, we have to answer two questions: (1) What constitutes a hyper-rule? In other words, how do we select the rules that form a hyper-rule? We address this question in Section 5.1. (2) How do we rank hyper-rules within a set of rules (either conventional or hyper)? We address this question in Section 5.2.

5.1 Hyper-Rule Formation

While in general any set of rules can be aggregated to form a hyper-rule, the focus of this paper is on the aggregation of rules that share a common consequent. These rules represent the collective impact of a change on the respective consequent, which then naturally forms the basis for a change recommendation.

Example 3. *Consider the set $R = \{\{a,b\} \rightarrow \{c,f\}, \{a,b\} \rightarrow \{c\}, \{d\} \rightarrow \{c\}\}$. For the purpose of change recommendation, we aggregate the last two rules in set R, since they share the same consequent. The antecedent of the resulting hyper-rule is the set of antecedents of all the constituent rules.*

Another potentially interesting application of rule aggregation is to aggregate rules with the same antecedent. Doing so facilitates determining the overall impact of a change. As an example, aggregating the first two rules in set R, introduced in Example 3, results in a hyper-rule that summarizes the impact of changing a and b together. The consequent of such a hyper-rule is the set of consequents of all the constituent rules.

Beyond these two, other problem domains may require still other ways of selecting rules for aggregation. In general, a hyper-rule, which intuitively summarizes a set of conventional rules, is defined as follows:

Definition 3 (Hyper-Rule). *Given a set of rules $R = \{A_1 \rightarrow C_1, ..., A_n \rightarrow C_n\}$ we define hyper-rule $\mathcal{H}(R)$ as*

$$\mathcal{H}(R) = \bigcup_{i=1}^{n} \{A_i\} \Rightarrow \bigcup_{i=1}^{n} \{C_i\}$$

Note that the antecedent and the consequent of a hyper-rule are sets of sets of entities rather than being sets of entities as found in conventional rules.

To help distinguish hyper-rules and conventional rules, we use a double-arrow $\Rightarrow$ in a hyper-rule rather than the single arrow $\rightarrow$ used with conventional rules.

Example 4. *Given R, the set of rules introduced in Example 3, let $R' \subset R$ be the set of rules that share the same consequent (i.e., $R' = \{\{a,b\} \rightarrow \{c\}, \{d\} \rightarrow \{c\}\}$). Then the hyper-rule generated from R' is:*

$$\mathcal{H}(R') = \{\{a,b\},\{d\}\} \Rightarrow \{\{c\}\}$$

Notice that the definition for a hyper rule concerns only the association rules and not the originating transactions. This opens up the possibility of identifying *cross transactional patterns*. In Example 4, the hyper rule simply states that when a and b change, c changes, and when d changes, c also changes. We do not require that all of a, b and d change together with c in the same transaction, rather we are only concerned with combining the evidence found in the individual association rules. Our hypothesis is that combining the evidence for c into a single *hyper rule* will better capture the collective evidence. The challenge here is to quantify the importance of a hyper rule, this we discuss in the next section.

5.2 Interestingness Measure Aggregation

In the same manner that an association rule has an interestingness measure a hyper-rule has an aggregated interestingness measure, which summarizes the interestingness values of all its constituent rules into a single value. Aggregated interestingness measures allow ranking hyper-rules; within a set of rules that may potentially contain both hyper-rules and conventional association rules. While an interestingness measure implies a total order over a set of conventional rules, an aggregated interestingness measure extends that total order to sets of rules that contain both hyper-rules and conventional association rules.

Although it is possible to define aggregated interestingness measures by extending each interestingness measure to be applicable to hyper-rules, a more scalable approach is to provide measure-agnostic aggregators that simply aggregate a number of interestingness values into a single value. Such aggregators should conform to a set of properties that are described in Hyper-Rule 4.

Definition 4 (Properties of a measure aggregator). *Let M be an interestingness measure defined over conventional rules, and R be a set of conventional rules. Let*

$\oplus$ denote a measure aggregator that maps $\mathcal{H}(R)$ and M to a single value representing the aggregated interestingness value (i.e., $\oplus(\mathcal{H}(R), M)$). Then the following properties should hold:

1. *Let r_1 and r_2 be two conventional rules, then*

$$M(r_1) \geq M(r_2) \implies \oplus(\mathcal{H}(\{r_1\}), M) \geq \oplus(\mathcal{H}(\{r_2\}), M)$$

2. *For each set of conventional rules R, if $|R| > 1$, then for each $r \in R$ the following holds:*

$$\oplus(\mathcal{H}(R), M) \begin{cases} > \oplus(\mathcal{H}(R - \{r\}), M) & if\ M(r) > 0 \\ = \oplus(\mathcal{H}(R - \{r\}), M) & if\ M(r) = 0 \\ < \oplus(\mathcal{H}(R - \{r\}), M) & if\ M(r) < 0 \end{cases}$$

The first property ensures that applying a measure aggregator, $\oplus$, on single rules retains their original ordering.

The second property ensures monotonicity. For example, it requires that the aggregation function is strictly increasing when rules with positive measure values are aggregated. Note that our theoretical framework for ranking hyper-rules is agnostic with regards to the interestingness measure used.

6 Aggregation Functions

In this section we present three aggregation functions that satisfy the properties of Properties of a measure aggregator 4 for *non-negative values*[3]. Thus this initial exploration of rule aggregators focus on aggregation of *positive correlation*, as we assess the inclusion of *negative correlation* to be a topic in it self. We briefly discuss potential strategies in subsection 6.5.

The first two aggregators, Cumulative Gain (CG) and Discounted Cumulative Gain (DCG), are adapted from well known performance measures in Information Retrieval. They are typically used to evaluate search results by evaluating a target list against an ideal (oracle) list [51]. In addition to these two, we introduce an additional aggregation function, Hyper Cumulative Gain (HCG). All aggregators are empirically evaluated in section 8.

6.1 Cumulative Gain

The first aggregation function, Cumulative Gain, comes from Information Retrieval [51].

[3]Formal proofs for the three aggregator functions are provided in the appendix.

Definition 5 (Cumulative Gain). *Given an interestingness measure M and a set of rules $R = \{r_1, \dots, r_n\}$, where $\forall r \in R\colon M(r) \geq 0$, the* Cumulative Gain of the hyper-rule $\mathcal{H}(R)$ is defined as follows:

$$CG\big(\mathcal{H}(R), M\big) = \sum_{i=1}^{n} M(r_i)$$

Example 5 (Cumulative Gain). *Given the following set of rules R, and an interestingness measure M:*

$$R = \{r_1, r_2, r_3\}$$
$$M = \{(r_1, 0.7), (r_2, 0.3), (r_3, 0.3)\}$$

the Cumulative Gain *of $\mathcal{H}(R)$ is computed as follows:*

$$CG\big(\mathcal{H}(R)\big) = 0.7 + 0.3 + 0.3 = 1.3$$

6.2 Discounted Cumulative Gain

While similar in nature to CG, DCG adds a coefficient that reduces the impact of subsequent values. This enables DCG to give greater weights to those values that have the largest impact [51]. Note that for DCG, internal ordering matters, in the following definition we therefore assume that rules are sorted from high to low according to their interestingness value.

Definition 6 (Discounted Cumulative Gain). *Given an interestingness measure M and a set of sorted rules $R = \{r_1, \dots, r_n\}$, where $\forall r \in R\colon M(r) \geq 0$, the DCG of the hyper-rule $\mathcal{H}(R)$ is defined as follows:*

$$DCG\big(\mathcal{H}(R), M\big) = \sum_{i=1}^{n} \frac{M(r_i)}{log_2(i+1)}$$

Notice that $\frac{M(r_i)}{log_2(i+1)}$ is monotonically decreasing because $log_2(i+1)$ is monotonically increasing while the rules values are decreasing. In subsection 11.1 we provide a formal proof that DCG satisfies the properties of Discounted Cumulative Gain 4 for non-negative values of M.

Example 6 (Discounted Cumulative Gain). *Consider the following set of rules R, with M giving the corresponding interestingness values:*

$$R = \{r_1, r_2, r_3\}$$
$$M = \{(r_1, 0.7), (r_2, 0.3), (r_3, 0.3)\}$$

DCG of $\mathcal{H}(R)$ for M is given by:

$$DCG(\mathcal{H}(R), M) = \frac{0.7}{log_2(2)} + \frac{0.3}{log_2(3)} + \frac{0.3}{log_2(4)} \approx 1.04$$

6.3 Hyper Cumulative Gain

The last aggregator function, *Hyper Cumulative Gain* (HCG), incorporates two properties, which makes it different from from CG and DCG. First, aggregation through HCG respects the bounds of the source interestingness measure. Second, it incorporates the *number of rules* that were aggregated to produce the hyper rule. In order to achieve this, HCG outputs a *pair* rather then a single value. We will talk about each element of the pair in order, and refer to them as HCG_1 and HCG_2.

HCG_1: Aggregating the interestingness measure values

For any given measure M, the values that HCG_1 generates are guaranteed to be within the range of M. This is as opposed to CG and DCG that do not necessarily preserve the original range of the respective interestingness measure. For example, the support measure has the range [0..1]. Given two support values of 0.8 and 0.7, the CG aggregator results in 1.5, which is greater than the upper bound 1. However, HCG_1 is 0.94, which is within the range [0..1]. Consequently, compared to CG and DCG, the aggregated values produced by HCG_1 put the hyper-rules on a more level playing field with the conventional rules, which are naturally constrained by their interestingness measure's range.

Apart from satisfying the range constraint, HCG_1 also has a probabilistic interpretation. From this perspective, HCG_1 provides the likelihood that at least one of the rules in a hyper-rule is relevant.

Example 7. *Let C be a change-set; r_1, r_2, and r_3 be three rules derived from C; and M be a measure that indicates the probability that a rule is relevant to its respective change-set. Then, the probability that at least one of r_1, r_2, or r_3 is relevant to C is calculated using the following formula:*

$$\begin{aligned} &M(r_1) + (1 - M(r_1)) * M(r_2) + (1 - M(r_1)) * (1 - M(r_2)) * M(r_3) \\ =&1 - (1 - M(r_1)) * (1 - M(r_2)) * (1 - M(r_3)) \end{aligned}$$

If a hyper-rule were composed of r_1, r_2, and r_3, its HCG_1 for measure M

would be calculated using the formula above. Generalizing this formula to an arbitrary number of rules, HCG$_1$ is defined as follows:

Definition 7 (HCG$_1$). *Given an interestingness measure M, with upper bound b, and a set of rules $R = \{r_1, \ldots, r_n\}$, where $\forall r \in R \colon M(r) \geq 0$, the HCG$_1$ of $\mathcal{H}(R)$ for M is defined as:*

$$HCG_1(\mathcal{H}(R), M) = M(r_1) + \sum_{i=2}^{n} \Big(M(r_i) \cdot \prod_{j=1}^{i-1} (1 - \frac{M(r_j)}{b}) \Big)$$

which, for finite values of b, is equivalent to

$$HCG_1(\mathcal{H}(R), M) = b - \prod_{i=1}^{n} (1 - \frac{M(r_i)}{b})$$

For measures without a finite upper bound ($b = \infty$), the term $\frac{M(r_j)}{b}$ is defined to be zero. In these cases, HCG$_1$ behaves the same as CG.

HCG$_2$: The number of rules

The second part of the HCG pair, HCG$_2$, captures the number of rules that were used to construct a hyper rule. From HCG$_1$ 4 we know that a rule which expresses no correlation through its interestingness measure value should not affect the aggregated value, HCG$_2$ satisfies this property by filtering out these rules. Following the filtering, HCG$_2$ is simply the cardinality of this filtered set:

Definition 8 (HCG$_2$). *Given a set of rules $R = \{r_1, \ldots, r_n\}$, HCG$_2$ of* R *is defined as:*

$$HCG_2(R, M) = |\{r \in R \mid M(r) > 0\}|$$

HCG: Combining HCG$_1$ and HCG$_2$

With HCG$_1$ and HCG$_2$ defined, HCG can simply be expressed as their ordered pair:

Definition 9 (Hyper Cumulative Gain). *Given an interestingness measure M, with upper bound b, and a set of rules $R = \{r_1, \ldots, r_n\}$, where $\forall r \in R \colon M(r) \geq 0$, the HCG of $\mathcal{H}(R)$ for M is defined as:*

$$HCG(\mathcal{H}(R), M) = \Bigg(HCG_1(\mathcal{H}(R), M),\ HCG_2(R, M) \Bigg)$$

To enable ranking hyper-rules based on their HCG values, we define the following total order relation over its value-count pairs.

Definition 10 (Total order relation over pairs). *For the pairs* (V_1, c_1) *and* (V_2, c_2) *the total order relation* $\geq$ *is defined as:*

$$(V_1, c_1) \geq (V_2, c_2) \equiv V_1 > V_2 \vee (V_1 = V_2 \wedge c_1 \geq c_2)$$

Note here that HCG_1 *takes precedence over* HCG_2*.*

Since HCG incorporates the range in its definition, we provide two examples, one for a measure that has a finite range, and one for a measure that has an infinite range.

Example 8 (Hyper Cumulative Gain over finite range). *Consider a set of rules R and an interestingness measure M with range* $[0, 1]$*:*

$$\begin{aligned} R &= \{r_1, r_2, r_3\} \\ M &= \{(r_1, 0.7), (r_2, 0.3), (r_3, 0.3)\} \end{aligned}$$

The HCG of $\mathcal{H}(R)$ *for M is given by:*

$$HCG(\mathcal{H}(R), M) = \Big(1 - (1 - 0.7) * (1 - 0.3) * (1 - 0.3), 3\Big) = (0.853, 3)$$

Example 9 (Hyper Cumulative Gain over infinite range). *Consider a set of rules R and an interestingness measure M with range* $[0, \infty)$*:*

$$\begin{aligned} R &= \{r_1, r_2, r_3\} \\ M &= \{(r_1, 0.7), (r_2, 0.3), (r_3, 0.3)\} \end{aligned}$$

Recall that the term $\frac{M(r_i)}{b}$ *is defined to be zero when the upper bound, b, is infinity; thus, HCG of* $\mathcal{H}(R)$ *for M is simply:*

$$HCG(\mathcal{H}(R), M) = \big(0.7 + 0.3 * (1 - 0) + 0.3 * (1 - 0)(1 - 0), 3\big) = (1.3, 3)$$

Notice that the HCG therefore is exactly equal to CG for interestingness measures with infinite max bound.

6.4 Centering

Most interestingness measures use the value 0 to indicate no correlation, one example is the original *support* measure. When the *support* of a rule, say $A \rightarrow B$, is equal to 0, the artifacts $a \in A$ and $b \in B$ have never all changed together in a single transaction. From the view of the *support* measure this is interpreted as there being no correlation between A and B. However, interestingness measures do not strictly need to use 0 as the point of no correlation, for the measures included in our study, *collective strength*, *lift*, and *odds ratio* are defined is such a way that *1* is the point of no correlation between the antecedent and consequent of a rule. Interestingness measures such as these must be re-centered.

Definition 11 (Centered Interestingness Measure). *An interestingness measure is* centered *if (1) its range contains the value 0; and (2) the value 0 indicates no correlation between the rule and the change set.*

To motivate the need for this centering, consider the following example:

Example 10 (Centering). *The lift interestingness measure has a range of* $[0..1..\infty)$ *with 1 indicating no correlation. Consider the following set of association rules, where the* lift *of each rule is also given:*

$$\begin{aligned} \{a\} &\rightarrow \{X\} \text{ lift: 1.5} \\ \{b\} &\rightarrow \{Y\} \text{ lift: 1} \\ \{c\} &\rightarrow \{Y\} \text{ lift: 1} \end{aligned}$$

Two of the rules share the same consequent and can therefore be aggregated. We do this twice, with and without centering.

not centered	centered
$\{\{b\},\{c\}\} \rightarrow \{Y\}\ CG([1,1]) = 2$	$\{a\} \rightarrow \{X\}\ CG(0.5) = 0.5$
$\{a\} \rightarrow \{X\}\ CG(1.5) = 1.5$	$\{\{b\},\{c\}\} \rightarrow \{Y\}\ CG([0,0]) = 0$

For the aggregated rule-set where lift *was not centered before aggregation, the two rules with Y as a consequent now rank higher than the X-rule,* even though there was no correlation between b, c, and Y. *However, if we center before aggregation, the* non-correlation is preserved after aggregation.

6.5 Aggregation of Negative Values

As stated earlier, our proposed aggregators are defined under the assumption that measure values are positive. Including negative values, correlations, in the aggregation complicates the situation in two ways:

1. Negative values may not have the same range as positive values for the same interestingness measure.

2. Depending on the aggregator, the order in which negative and positive values are mixed can have significant implications.

One possible remedy for (1) may be some sort of normalization, while for (2) we envision that negative and positive may be aggregated separately, using the absolute value of the negative value. However, given that our study incorporates few interestingness measures that empirically produce negative values, we have left this exploration for future work.

7 Experiment Design

To assess the viability of association rule aggregation and especially the measurement aggregation functions proposed in section 6, we perform a large-scale empirical study. While we believe that association rule aggregation will be useful in a variety of problem domains, our study focuses on change recommendation. In other words, we focus on aggregating rules that share the same consequent. This is because, as discussed in section 5, only hyper rules of this form establish the basis for a change recommendation.

The evaluation investigates the performance of association rule aggregation when using different aggregation functions, in the context of various software-systems and various interestingness measures. Furthermore, it investigates if and how the *granularity* of the underlying change history affects the result of association rule aggregation. Two granularity levels are considered: file level and method level. Specifically, the following questions are investigated:

RQ 1. *How frequently can change recommendation be improved by association rule aggregation?*

RQ 2. *What is the effect of aggregating association rules for change recommendation?*

RQ 3. *What is the effect of aggregating association rules within each studied software-system?*

RQ 4. *How much does a change in granularity impact the precision gain from rule aggregation?*

In total, we generated approximately 21.8 million data points to answer our four research questions. The remainder of this section will describe our study in detail. To start, Figure B.1 provides a high-level overview of the experiment design.

7.1 Subject Systems

To assess the impact of association rule aggregation under a range of conditions, we study 17 large systems with varying characteristics, such as frequency of changes, number of file and method changes, and average number

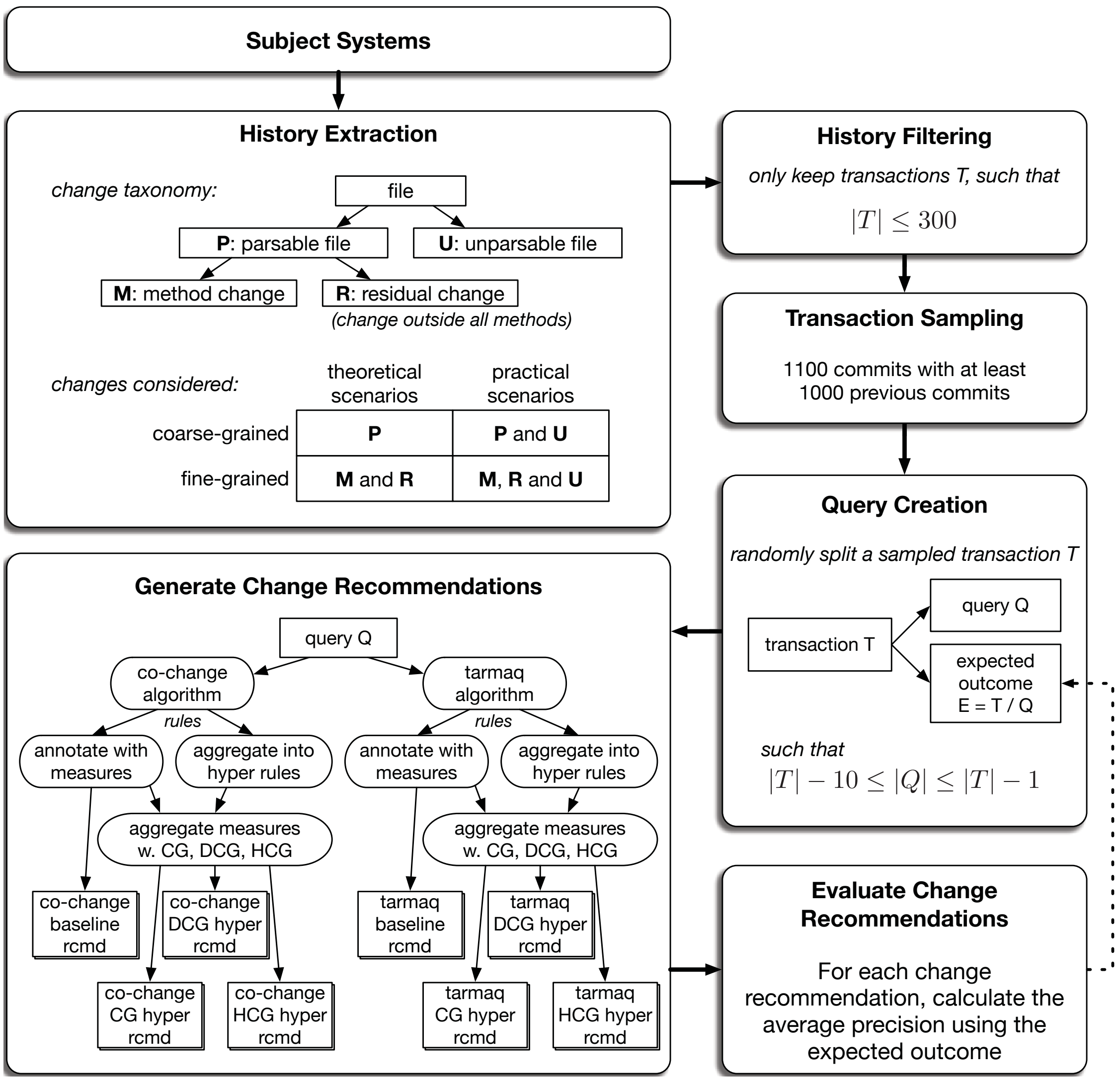


Fig. B.1: The high level flow of our experiment design

of changes per commit. Two of the systems come from our industry partners, Cisco Norway and Kongsberg Maritime (KM). Cisco Norway is the Norwegian division of Cisco Systems, a worldwide leader in the production of networking equipment. We analyze their software product line for professional video conferencing systems. KM is a leader in the production of systems for positioning, surveying, navigation, and automation of merchant vessels and offshore installations. We analyze the common software platform that KM uses across various systems in the maritime and energy domain.

The other 15 systems include the well known open-source projects shown in the first column of Table B.3. In addition to information regarding the extracted histories (discussed in subsection 7.2), the table shows that the systems vary from medium to large size, ranging up to just over 280 000 unique files in the largest system. Finally, the lower subtable shows the programming languages used in each system, as an indication of heterogeneity.

7.2 History Extraction

From each of the 17 subject systems, we extract *four* different histories based on up to the 50 000 most recent transactions (*commits*). This choice is motivated by our previous work in the area of software repository mining [52], which suggests that considering such a number of transactions does not include outdated co-change information. The four differ in terms of *granularity* and *parsability*. Here granularity is either file level or method level, and parsability either includes or excludes files that can be parsed into methods. Parsability is tied back to our history extractor's use of SrcML [53], which supports method-level parsing of C, C++, C#, and Java code. In addition to these languages the change histories of our subject systems contain code written in a multitude of other languages such as Python, Ruby, and JavaScript (as well as build/configuration files in XML, etc.).

From the point of view of a developer in need of a change recommendation, the more fine-grained the response the better. For example, it is easier to act on the recommendation "consider changing method M", than the recommendation "consider changing file F". On the other hand, there exist evolutionary couplings between files where method-level change information is unavailable. In such cases file-level recommendations are of more use than no recommendations at all.

We explore the impact of granularity under two different scenarios: *the practical scenario* and *the theoretical scenario*. Under the practical scenario we acknowledge that there are many files for which we do not have method-level information and include file-level changes in such cases. In contrast, under the theoretical scenario we study the hypothetical situation in which we have method-level data for all changes. Because we, in fact, do not have such information, we approximate this situation by removing from the analysis

Table B.3: Characteristics of the evaluated software systems (based on our extraction of the last 50 000 transactions for each of he systems).

Software System	History (in yrs)	Unique # files	Unique # artifacts	Avg. # artifacts in commit
CPython	12.05	7725	30090	4.52
Mozilla Gecko	1.08	86650	231850	12.28
Git	11.02	3753	17716	3.13
Apache Hadoop	6.91	24607	272902	47.79
HTTPD	19.78	10019	29216	6.99
Liferay Portal	0.87	144792	767955	29.9
Linux Kernel	0.77	26412	161022	5.5
MediaWiki	9.87	12252	12252	5.43
MySQL	10.68	42589	136925	10.66
PHP	10.82	21295	53510	6.74
Ruby on Rails	11.42	10631	10631	2.56
RavenDB	8.59	29245	47403	8.27
Subversion	14.03	6559	46136	6.36
WebKit	3.33	281898	397850	18.12
Wine	6.6	8234	126177	6.68
Cisco Norway	2.43	64974	251321	13.62
Kongsberg Maritime	15.97	35111	35111	5.08

Software System	Languages used*
CPython	Python (53%), C (36%), 16 other (11%)
Mozilla Gecko	C++ (37%), C (17%), JavaScript (21%), 34 other (25%)
Git	C (45%), shell script (35%), Perl (9%), 14 other (11%)
Apache Hadoop	Java (65%), XML (31%), 10 other (4%)
HTTPD	XML (56%), C (32%), Forth (8%), 19 other (4%)
Liferay Portal	Java (71%), XML (23%), 12 other (4%)
Linux Kernel	C (94%), 16 other (6%)
MediaWiki	PHP (78%), JavaScript (17%), 11 other (5%)
MySQL	C++ (57%), C (18%), JavaScript (16%), 24 other (9%)
PHP	C (59%), PHP (13%), XML (8%), 24 other (20%)
Ruby on Rails	Ruby (98%), 6 other (2%)
RavenDB	C# (52%), JavaScript (27%), XML (16%), 12 other (5%)
Subversion	C (61%), Python (19%), C++ (7%), 15 other (13%)
WebKit	HTML (29%), JavaScript (30%), C++ (26%), 23 other (15%)
Wine	C (97%), 16 other (3%)
Cisco Norway	C++, C, C#, Python, Java, XML, other build/config
Kongsberg Maritime	C++, C, XML, other build/config

* languages used by open source systems are from `http://www.openhub.net`, percentages for the industrial systems are not disclosed.

files for which we don't have method-level data. Thus, we extract four change histories divided into two distinct classes, practical and theoretical. These histories are formalized as follows:

Practical Scenarios The two practical scenarios capture the reality that parsing all files is infeasible. In this case we compare a pure file-level history, with a mixed history that incorporates as much method-level information as possible. The two histories used for the practical study are:

practical coarse-grained: A history where each transaction includes the files changed in the respective commit. The *practical coarse-grained* histories recieve the least processing. They are essentially what is returned by `‘git log‘` after filtering as described in the next section.

practical fine-grained: A history where each *parseable* file of *practical coarse-grained* is replaced by the changed methods of the file together with the file's "*residual*," which is included only if there are changes to source-code outside all of the file's methods.

Theoretical Scenarios The theoretical scenario is used to explore the question "what would happen if all files were parseable?". As discussed above, this theoretical ideal is achieved by pruning the data. It considers the following histories:

theoretical fine-grained: A history that discards from *practical fine-grained* all unparseable files.

theoretical coarse-grained: A history that discards from *practical coarse-grained* all unparseable files (thus *theoretical coarse-grained* is *theoretical fine-grained* projected back to the file level).

Table B.3 reports statistics related to the extracted change histories, which illustrate the diversity of the systems studied. For example, the transactions cover vastly different time spans, ranging from almost 20 years in the case of HTTPD, to a little over 10 months in the case of the Linux kernel. The table also shows the number of unique files changed in the *practical coarse-grained* data set, as well as the number of unique artifacts changed in the *practical fine-grained* data set.

We now provide an example of how a single commit is interpreted in the four different histories:

Example 11 (History Parsing). *Consider the following changes to the files `A.c`, `B.cpp` and `C.yaml`, which were all added to the same commit. First, in the C file `A.c`, a line was changed in the method `m1(int p)`. Secondly, in the C++ file `B.cpp`, a line was changed in the method `m2(int p)` and a public variable was* *changed in the parent class. Lastly, in the configuration file* `C.yaml`*, a single line was changed.*

A.c

```
  int m1(int p) {
-   old_line
+   new_line
  }
```

B.cpp

```
  class Class1 {
-   public var_old
+   public var_new

    int m2(int p) {
-     old_line
+     new_line
    }
  }
```

C.yaml

```
-   old_config: X
+   new_config: Y
```

The changes of these three files, found in the same commit, are interpreted in four different ways to construct four different transactions. We will now list how each of the four types of histories studied in this paper, represent the changes found in the commit. Note that changes which occur outside of all methods are tagged as `file:@residuals`*.*

practical coarse-grained *{*`A.c`*,* `B.cpp`*,* `C.yaml`*}*
: *All changed files found in the commit are included as is in the transaction.*

practical fine-grained *{*`A.c:m1`*,* `B.cpp:m2`*,* `B.cpp:@residuals`*,* `C.yaml`*}*
: *C and C++ is supported for method level parsing, so fine-grained information is included for those files. The yaml file is not supported and is included as is.*

theoretical coarse-grained *{*`A.c`*,* `B.cpp`*}*
: *All changed files which are supported for fine-grained parsing are included.*

theoretical fine-grained *{*`A.c:m1`*,* `B.cpp:m2`*,* `B.cpp:@residuals`*}*
: *Fine-grained information from the parseable files is included, the yaml file hence is discarded.*

Note that class and parameter information also is encoded on artifacts to deal with name overloading, but this information is left out of the example above for readability.

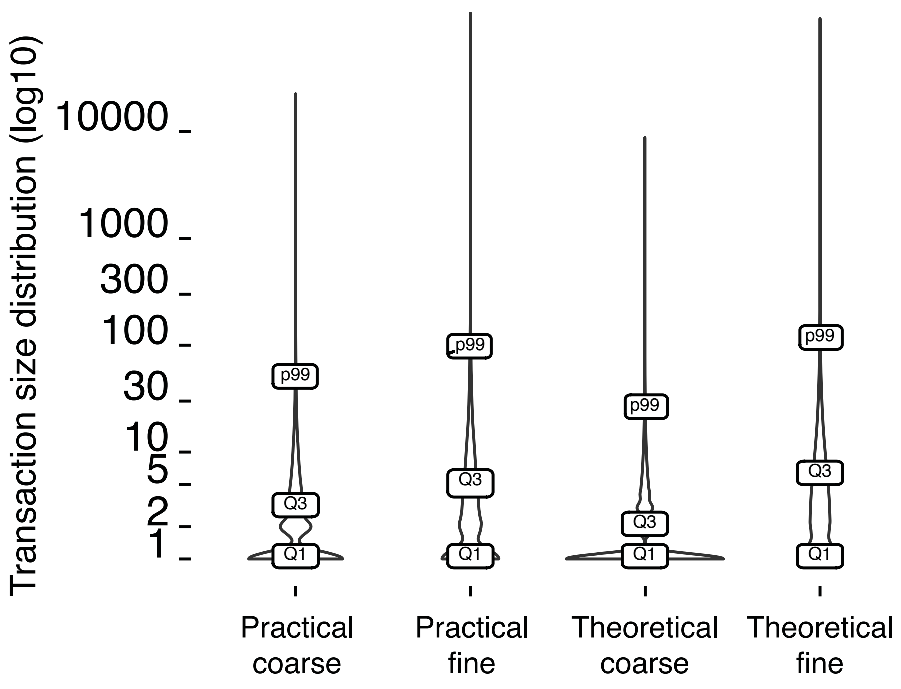


Fig. B.2: An overview of the overall distributions across all subject systems for each granularity. Each distribution is shown annotated with the three quartiles, as well as the 99% percentile (for some distributions the percentiles overlap, in those cases the larger percentile is on top).

7.3 History Filtering

One challenge faced by association rule mining is that large transactions lead to a combinatorial explosion in number of association rules [12]. Fortunately, as seen in Figure B.2, which provides violin plots of transaction size for the four data sets, transaction sizes are heavily skewed towards smaller transactions. This pattern is consistent across the individual systems. For example, using the *practical fine-grained* histories Figure B.3 provides separate violin plots for each system.

Unfortunately, as also seen in the violin plots, there exist outlier transactions containing 10 000 or more artifacts. To combat the combinatorial explosion problem raised by such large commits, it is common to filter the history. In an attempt to reflect *most* change recommendation scenarios, we employ a quite liberal filtering and remove only those transactions larger than *300* artifacts. The rational behind choosing this cutoff is that for each program at least 99% of all transactions are smaller then 300 artifacts. In most cases, the percentage is well above 99% of the available data.

7.4 Transaction Sampling and Query Creation

Conceptually, a *query* Q represents a set of files that a developer changed since the last synchronization with the version control system. The key idea

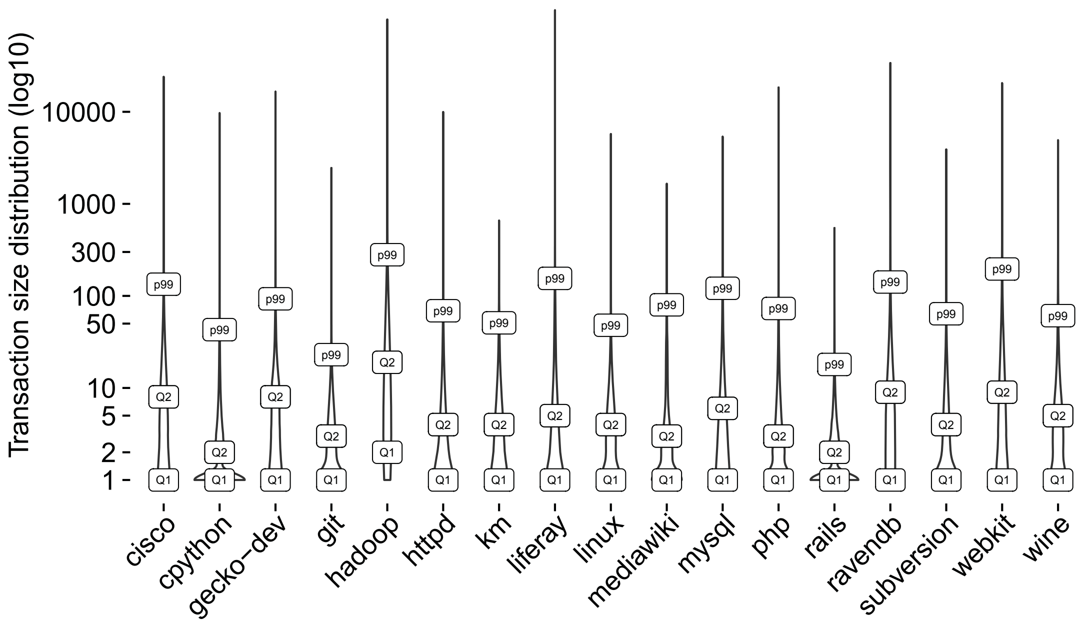


Fig. B.3: An overview of the distributions of transactions sizes for each subject system (*practical fine-grained* history).

behind our evaluation is to generate from each sampled transaction T, a query that emulates a developer errantly forgetting to update some subset of T. To this end, we partition each transaction T into a non-empty query Q and a non-empty expected outcome $E \overset{\text{def}}{=} T \setminus Q$. In this way, we can evaluate the ability of a recommendation tool to infer E from Q.

From each filtered history we take a representative sample of 1100 transactions,[4] with the following two constraints:

- The transaction must contain at least three artifacts. This constraint ensures that, at the minimum, a transaction can be split into a query of at least two artifacts and an expected outcome of at least one. Two artifacts are the minimum for there to be the possibility that a recommendation will contain at least two rules that can be aggregated.

- The transaction must have a previous history of at least 1000 trans-

[4]For a normally distributed population of 50 000, a minimum of 657 samples is required to attain 99% confidence with a 5% confidence interval that the sampled transactions are representative of the population. Since we do not know the distribution of transactions, we correct the sample size to the number needed for a non-parametric test to have the same ability to reject the null hypothesis. This correction is done using the Asymptotic Relative Efficiency (ARE). As AREs differ for various non-parametric tests, we choose the lowest coefficient, 0.637, yielding a conservative minimum sample size of $657/0.637 = 1032$ transactions. Hence, a sample size of 1100 is more than sufficient to attain 99% confidence with a 5% confidence interval that the samples are representative of the population.

actions. This constraint ensures that there is a minimum number of transactions in the training set for the mining algorithms.

7.5 Generate Change Recommendations

All queries are executed using two different targeted association rule mining algorithms, namely TARMAQ and CO-CHANGE (introduced in section 2). Executing a query Q, created from a transaction T, creates a set of association rules. The rules often differ based on the algorithm used. Moving from a set of rules to a change-recommendation with respect to Q, requires giving weight to the rules such that they can be sorted. In this paper we experiment with the 40 interestingness measures shown in Table B.2.

Central to this paper, and as first envisioned in section 3, there exists a potential to improve the recommendation by combining the evidence captured in the individual association rules. We explore this potential by aggregating rules that share the same consequent into a hyper rule, and weighing it using the measure aggregators presented in section 6. For any *one* query, we therefore create four different recommendations: the original recommendation, and three recommendations produced by aggregating the rules of the original recommendation using the aggregators CD, DCG, and HCG.

7.6 Evaluate Change Recommendations

To evaluate each recommendation we compute its *average precision* (AP). This value captures the precision computed at each relevant document (i.e., each expected outcome) and thus favors recommendations where relevant documents are toward the beginning of the list. Furthermore, we capture the performance over a set of queries (e.g., when using one of the two rule-generation algorithms with a given interestingness measure) using the *mean average precision* (MAP). Formally average precision is defined as follows:

Definition 12 (Average Precision). *Given a recommendation* R*, and an expected outcome* E*, the average precision of* R *is given by:*

$$AP(R) \overset{def}{=} \sum_{k=1}^{|R|} P(k) * \triangle r(k)$$

where $P(k)$ is the precision calculated on the first k files in the list (i.e., the fraction of correct files in the top k files), and $\triangle r(k)$ is the change in recall *calculated only on the $k-1^{th}$ and k^{th} files (i.e., how many more correct files where predicted compared to the previous rank).*

Note that since we consider only rules with singleton consequents, $\triangle r(k)$ will always be equal to either zero or $1/|E|$ (i.e., a rank either does not contain a file from the expected outcome, or it contains exactly one file from the expected outcome). Table B.4 illustrates the computation of AP, $P(k)$, and $\triangle r(k)$ given the ranked list $[c, a, f, g, d]$ and the expected outcome $\{c, d, f\}$.

8 Results and Discussion

This section presents the results of the study described in section 7, and is structured according to our four research questions: We first discuss RQ 1 in subsection 8.1 on how often our technique for rule aggregation can be applied. This is followed by RQ 2 in subsection 8.2 on how aggregation may improve change recommendation. In subsection 8.3 we discuss RQ 3, which considers aggregation performance over individual software systems, and finally in subsection 8.4 we discuss RQ 4, which explores the effect of artifact granularity in the context of aggregation. The first three research questions consider patterns *within* a single history, here the *practical fine-grained* history is used because it captures the largest number of changed artifacts at the finest level of granularity, making it the most useful in real world change recommendation scenarios. However, we did repeat the analysis of RQ 1, RQ 2 and RQ 3 using the other histories and found effectively the same patterns.

The final research question considers all four histories. It explores the relative performance of association rule aggregation as a function of granularity and parsability.

8.1 Applicability of Hyper-Rules (RQ 1)

As discussed in section 3, a recommendation can be aggregated if there are at least two rules which share the same consequent, in this section we investigate how often this scenario occurs in practice.

Table B.4: Calculation of average precision, based on ranked list $[c, a, f, g, d]$ and expected outcome $\{c, d, f\}$

Rank (k)	Artifact	$P(k)$	$\triangle r(k)$
1	c	1/1	1/3
2	a	1/2	0
3	f	2/3	1/3
4	g	2/4	0
5	d	3/5	1/3

$$\textit{average precision } (AP) = \\ 1/1 * 1/3 + 1/2 * 0 + 2/3 * 1/3 + 2/4 * 0 + 3/5 * 1/3 \approx 0.75$$

As explained in section 2, the two algorithms used in our study, Co-Change and Tarmaq, sit at opposing ends with respect to their approach to rule generation. Co-Change on the one hand, splits the input query into its individual artifacts, and generates all possible singleton rules for each artifact. Doing so increases the odds that multiple rules will share the same consequent. On the other hand, the antecedents found in rules mined by Tarmaq are dynamically determined by searching for the largest subset of the query that has some support in the history. More often than not, these subsets are close to the query, resulting in less variation in unique antecedents, and therefore also less possibility for rules which share the same consequent.

To analyze the effect that the choice of association rule mining algorithm has on applicability of hyper-rules, we count the number of recommendations that contain aggregable rules. The expectation here is that Co-Change, by virtue of its creating the maximal amount of rules given a query, also will produce recommendations which are frequently aggregable. The experiment bears out this expectation. For the *practical fine-grained* history the recommendations generated by Co-Change were aggregable 84% of the time, while the recommendations generated by Tarmaq were aggregable only 15% of the time.

8.2 Ability to Improve Precision (RQ 2)

The results for RQ 1 show that there are ample of opportunities for association rule aggregation. RQ 2 considers the impact aggregation has on the quality of the resulting recommendation. To address RQ 2, the evaluation considers three dimensions: the rule generation algorithm used (Co-Change or Tarmaq), the interestingness measure used to rank the rules, and the aggregation function used to form the hyper-rules. Hereafter we refer to an (algorithm, measure) combination as a *case*. We report the result of a statistical comparison of the mean average precision and two measures of effect size. To test for statistically significant differences between original and aggregated recommendations we use a one-tailed, paired Wilcoxon signed rank tests. For each case we compare the recommendations without aggregation against each of the aggregated recommendations.

The first of the two effect-size measures is the standardized effect size, which is calculated by dividing the Wilcoxon p-value's corresponding z-statistic by the square root of the number of observations to obtain *Pearson's r* [54].

$$(Pearson)\ r = \frac{z}{\sqrt{N}}$$

The second measure of effect size is the non-standardized effect size, which

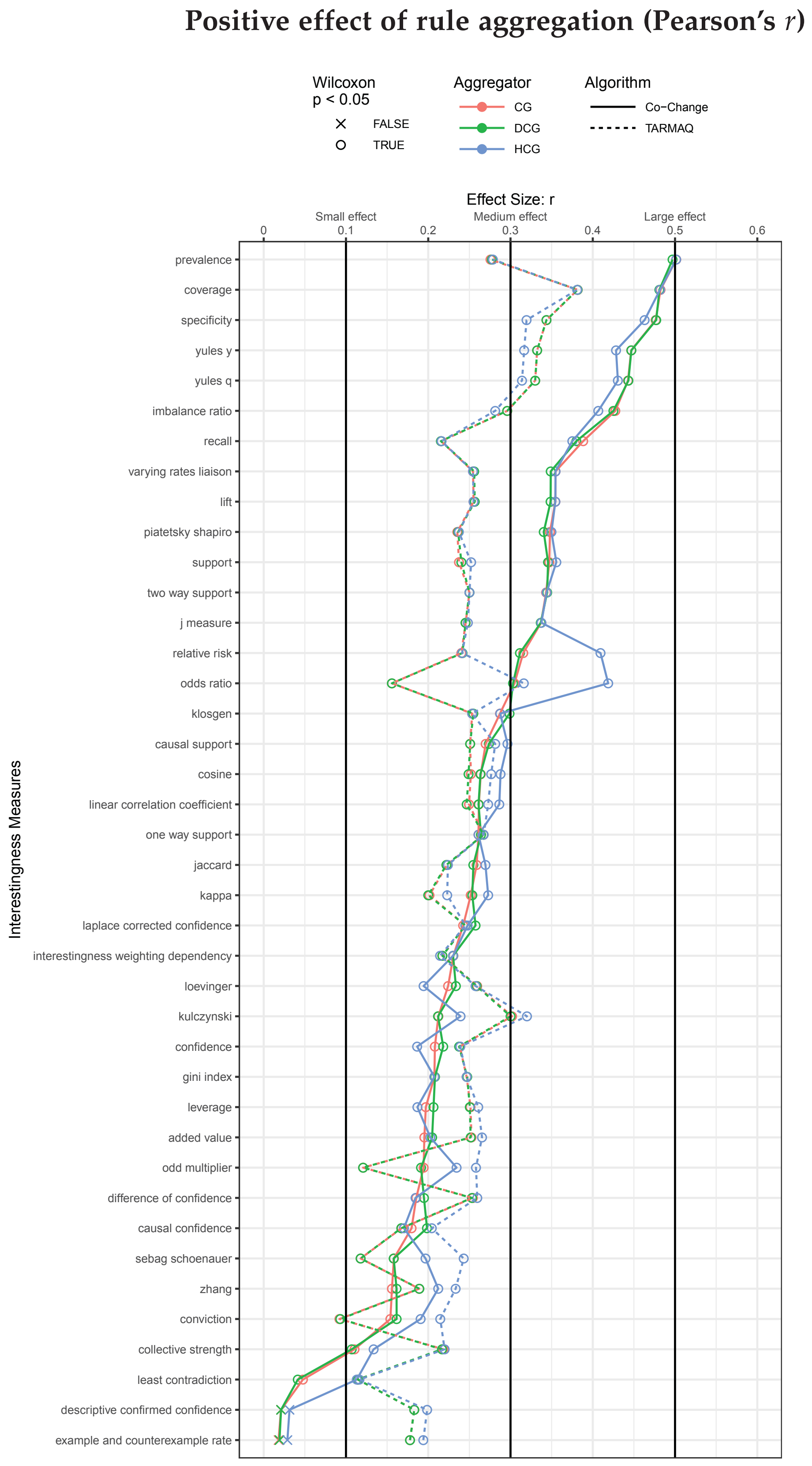


Fig. B.4: Effect of aggregating change recommendations using various combinations of interestingness measures, mining algorithms and aggregation functions. Effect size given by Pearson's r.

is defined as the *percentage change in mean average precision* (CiMAP):

$$(\%)\ CiMAP = \frac{MAP_{aggregated} - MAP_{original}}{MAP_{original}} * 100$$

The p-values and standardized effect sizes are shown in Figure B.4 while the non-standardized effect sizes are shown in Figure B.6. In both figures the interestingness measures are ordered based on the respective y-measure (r or CiMAP) for CG over Co-Change recommendations, note that this choice is incidental and is done purely to ease later comparisons.

Figure B.4 shows both the p-values and the corresponding effect size measured, r. The results of the Wilcoxon tests are shown using *hollow circles* and *crosses* where a circle designates a significant result ($p < 0.05$) and a cross a non-significant result. To ease the effect size interpretation, vertical black bars have been added at r = 0.1, 0.3 and 0.5, corresponding to what typically are considered small, medium, and large effects [55]. The effect of aggregating Co-Change recommendations is shown with solid lines, while Tarmaq is shown with dotted lines. Furthermore, color is used to differentiate the three aggregators with CG shown in red, DCG in green, and HCG in blue.

In all cases aggregation has a positive effect on the change recommendation, meaning that the average precision of the aggregated recommendations tend to be higher than that of their non-aggregated counterparts. This effect is significant in all but two cases: *example and counterexample rate* and *descriptive confirmed confidence* using Co-Change. In terms of the size of the effect, aggregation of Tarmaq recommendations results in relatively stable low to medium positive effects across most interestingness measures. In the case of Co-Change there is larger spread in effect sizes with a few measures experiencing no to low effect, while the remaining measures are split between experiencing a low to medium or a medium to large effect.

As an overall positive effect has been found, we now turn to a direct comparison between the aggregators. First observe that CG and DCG closely follow each other across Co-Change and Tarmaq and across all the measures, this indicates that their recommendations tend to be quite similar. Thus in practice either one can be used, although we would recommend CG as its aggregated values are easier to interpret.

The more interesting comparison is between CG/DCG and HCG, where HCG seems to perform significantly better on a handful of interestingness measures. To investigate the differences more closely we applied the Wilcoxon test to just CG and HCG. The results are shown in Figure B.5. Here the interestingness measures have been partitioned based on their range. Recall that HCG incorporates the range into its definition. In Figure B.5, the bottom three partitions contain the interestingness measures with finite ranges, while the top three contain those with infinite ranges. For the interestingness measures with finite ranges, HCG was not significantly better than CG with the exception of two cases (*laplace corrected confidence* and *descriptive confirmed confidence*). Furthermore, for these cases the effect was small. On the other hand, HCG is much better for interestingness measures with infinite range. Recall that the way HCG is defined, aggregated values for CG and HCG are the same when the range is infinite; however, HCG also incorporates a tie breaking mechanism based on the number of rules aggregated to create a hyper rule. Thus the only difference between HCG and CG for the infinite range measures is this tie breaker. It is evident that tie breaking brings a positive effect on aggregation. Given these findings we would recommend the CG aggregator in combination with the tie breaking mechanism of HCG. In simpler terms, a promising rule aggregator is simply one that *sums the respective interestingness measure values* and breaks ties by *the number of values in the sum*.

Improvement measured by change in MAP

So far we have discussed the effect of aggregation in terms of a standardized measure of effect size, however, this comes at the cost of distancing the measure from the original measurement unit which is *average precision*. As a complementary view of the data we consider CiMAP, which captures difference in MAP when recommendations are aggregated. The data is shown in Figure B.6, which again has the interestingness measures sorted based on CG performance using the Co-Change recommendation.

Comparing the two, the top five measures when using the standardized measure (Figure B.4) were *prevalence*, *coverage*, *specificity*, *yules y* and *yules q*. There is a large overlap with the non-standardized measure: four of five measures are the same. The only difference is that *prevalence* is ranked lower when using CiMAP. In Figure B.4 we found that *coverage* experienced a medium to large effect of aggregation for Tarmaq and a large effect for Co-Change. We can now see how this effect size translates to one using average precision; the MAP of aggregated Tarmaq recommendations improved by a factor of approximately *2.5* (a 150% increase), while the MAP of aggregated Co-Change recommendations improve by a factor of approximately *3* (a 200% increase).

Implications of Results

Looking beyond RQ 2, our results also support several other interesting observations. First, we found that both Co-Change and Tarmaq produced recommendations which responded well to aggregation. This should be emphasized, as the two algorithms significantly differ in their way of rule generation; Co-Change can be thought of as generating the maximum number of rules, while Tarmaq can be thought of as generating the minimum amount of high relevance rules. From this we posit that there is a high likelihood that

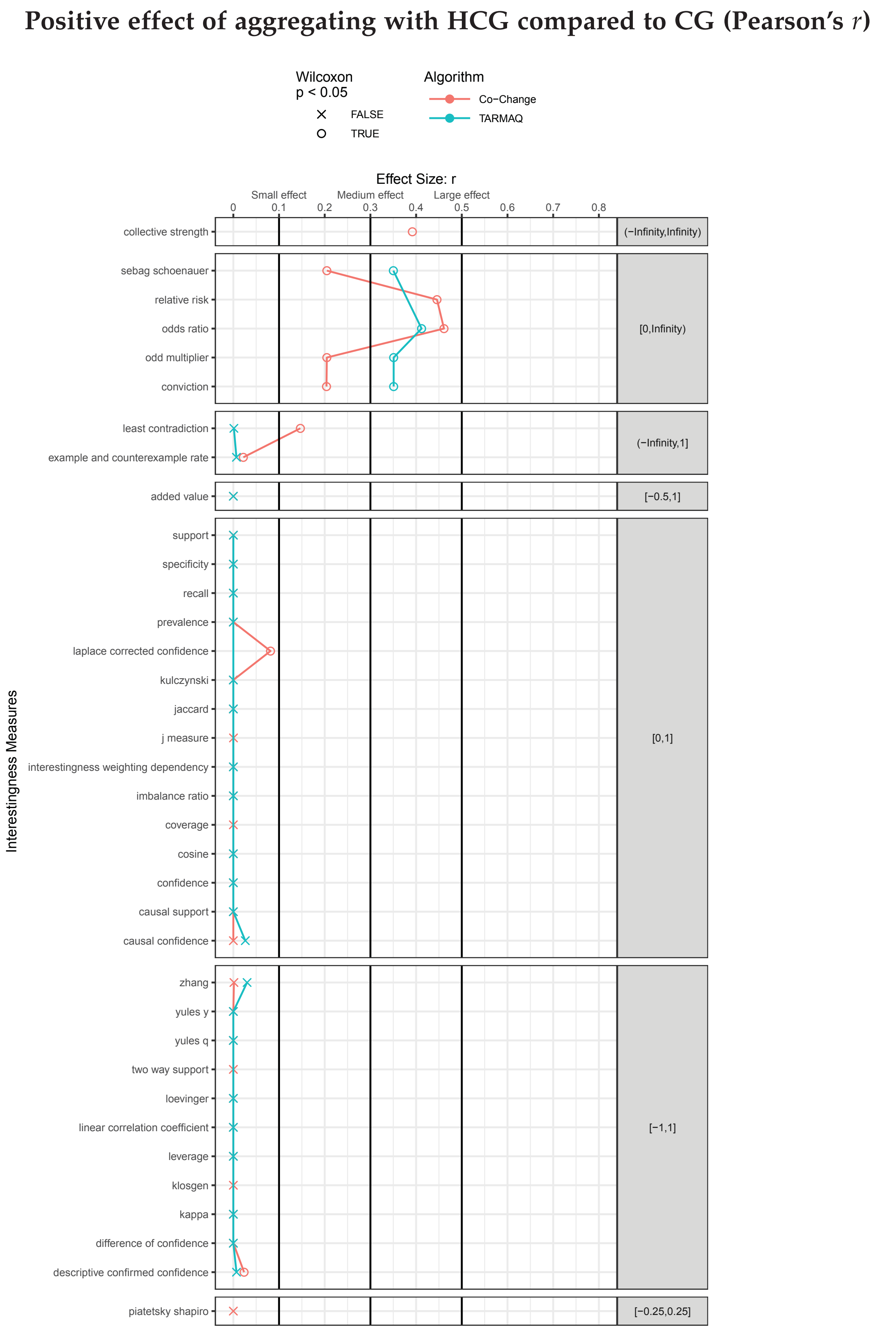


Fig. B.5: Positive effect of aggregating rules using the HCG aggregator compared to using the CG aggregator. Interestingness measures are grouped based on their range.

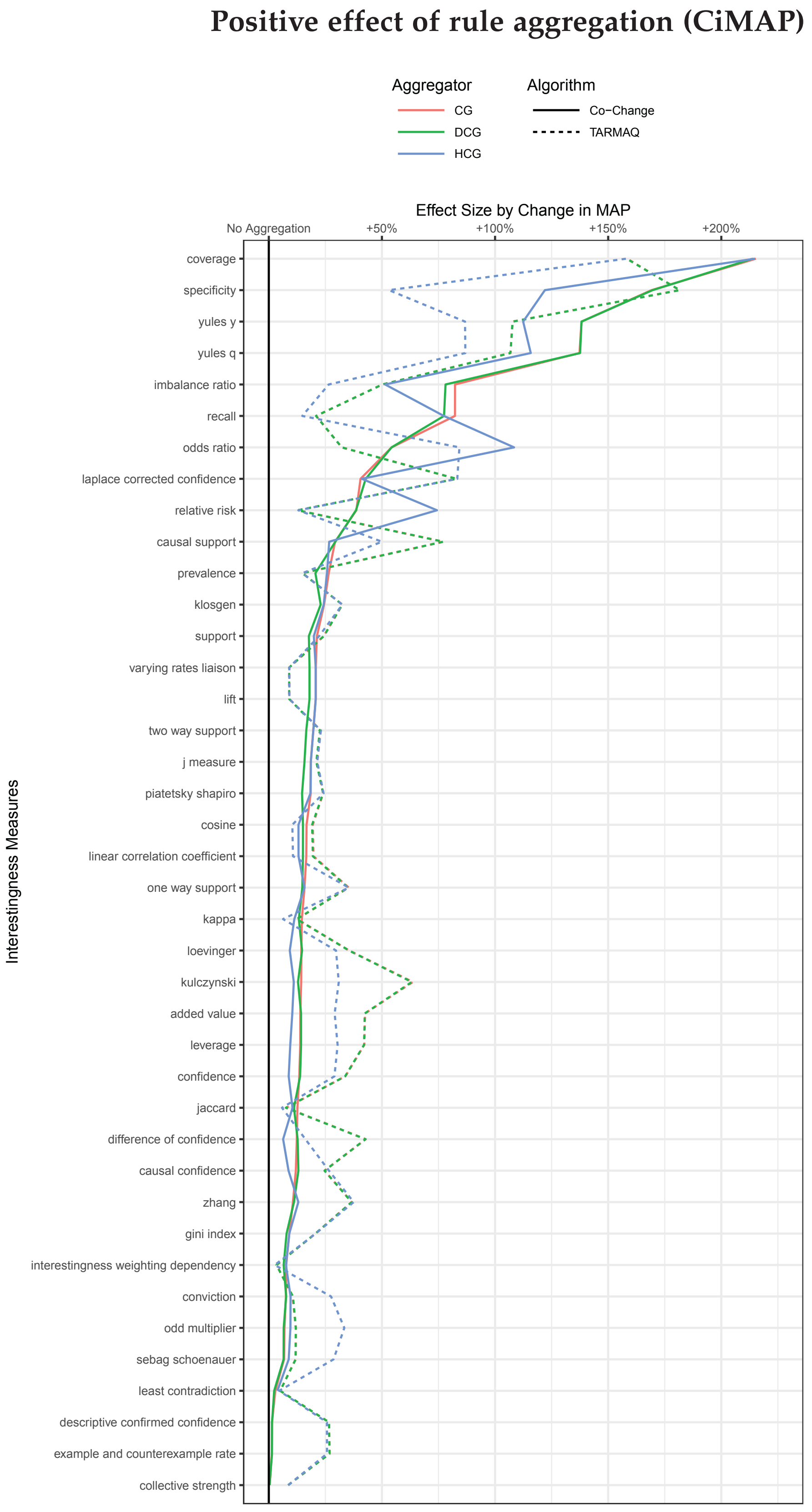


Fig. B.6: Effect of aggregating change recommendations using various combinations of interestingness measures, mining algorithms and aggregation functions. Effect size given by *change in mean average precision (CiMAP)*.

also other association rule mining algorithms will benefit from rule aggregation.

Turning to the three studied aggregation functions, our results can be used to select an appropriate aggregation function given the algorithm and interestingness measures which one is desirable to use. However, from a practical standpoint, more often than not the differences between aggregators are minimal. As an explanation of this consider Table B.5. As we can observe, over 99% of all hyper-rules are created from two to three rules in the case of Tarmaq, while the same number for Co-Change is approximately 72%. As the aggregators of section 6 essentially only differ in their coefficients in a sum, the more rules which are aggregated, the larger the differences are between the aggregated values. So, as typically only 2 or 3 rules are used to form hyper-rules, differences between the values produced by the different aggregators are also minimized, making the final recommendations also similar in nature. Looking forward, an interesting venue might be to theorize aggregation functions which maximizes the benefit from only aggregating a few rules.

Table B.5: The mean number of rules used to form each hyper-rule.

	[2,3)	[3,4)	[4,5)	[5,6)	[6,7)	[7,8)	[8,9)	[9,10)	10+
Co-Change	71.69%	11.61%	5.22%	2.65%	1.88%	1.02%	0.98%	0.72%	4.20%
Tarmaq	99.6%	0.32%	*4+ = 0.08%*						

8.3 System Specific Analysis (RQ 3)

In prior sections we have explored overall trends which emerge when association rules are aggregated. However, as the studied software systems (see Table B.3) differ on several aspects such as languages used, frequency/size of commits, etc., we consider the possibility that the aggregators might exhibit system-specific differences. To investigate this potential we analyze the effect of aggregating recommendations generated for each individual software system using the same method (Wilcoxon and Pearson's r) as for the system-agnostic analysis in RQ 2. The results can be found in the six plots shown in Figure B.7, where each plot provides the result for one combination of algorithm and aggregator. Furthermore, the software systems of each plot are ordered based on the mean effect size within each system (i.e., the software system on top saw the most benefit from aggregation). As reference points we have highlighted a selection of interestingness measures from the system-agnostic analysis.

We first consider the observed range of effect sizes for the system-specific analysis. In our system-agnostic analysis we found that aggregation of Co-

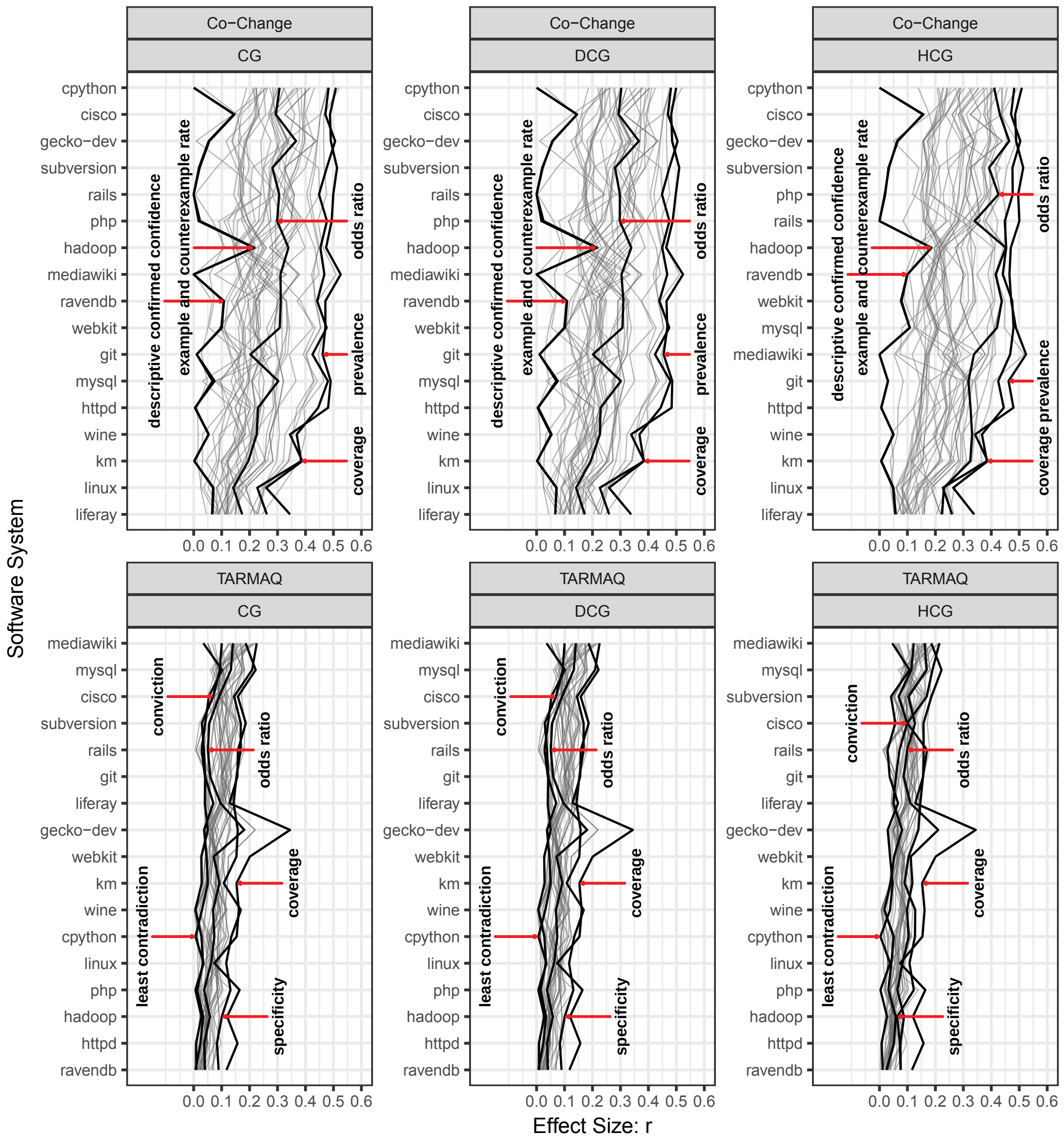


Fig. B.7: Effect of aggregating interestingness measures across software systems. A selection of high and low performing interestingness measures from the system-agnostic analysis are highlighted (see Figure B.4).

Change recommendations resulted in a wide range of effect sizes from essentially *no effect* to a *large effect*. In the system-specific analysis we find the same pattern. For Tarmaq we also see the same effect size range. However, there clearly also exist differences between software systems. In particular, *Linux* and *liferay* experience less benifit from aggregation with Co-Change recommendations. For interestingness measures that saw very little effect in the system-agnostic study, there also exist specific software systems where these result in medium to large effects, one example is *example and counterexample rate* for *hadoop*. In the case of Tarmaq, the *coverage* interestingness measure experiences a large effect for *gecko-dev* but only a small effect with the other software systems.

We will now turn to the *labeled* interestingness measures of Figure B.7. We first consider the interestingness measures that showed little effect from aggregation in the system-agnostic study. For these interestingness measures we can observe the same pattern for the system-specific results; for Co-Change the *descriptive confirmed confidence* and *example and counterexample rate* consistently has the least effect, while for Tarmaq the *least contradiction* also consistently has the least effect across all systems. Furthermore, Tarmaq with *conviction*, for which we previously saw a considerably larger effect using HCG compared to CG/DCG, can now be connected to the specific software systems where HCG outperforms the other aggregators (e.g., cisco, rails, git). If we turn to the interestingness measures where aggregation had a large effect, those same measures also see the largest effect of aggregation when observed on the software-system level (*prevalence*, *coverage*, *specificity*). In addition to the extremities discussed so far, we have also highlighted *odds ratio* because of its diverse results in the system-agnostic analysis. For Co-Change and CG/DCG, odds ratio saw average effect of aggregation relative to other measures, while it had one of the largest effects for HCG. Interestingly we see the same pattern reflected in the system-specific results of Figure B.7. A similar pattern is reflected in the results for Tarmaq.

In summary the effect of aggregation on interestingness measures to a large degree are consistent across software systems. However, some interestingness measures are more prone to system specific deviations and should therefore be evaluated on a case by case basis.

8.4 Effect of Granularity (RQ 4)

So far we have considered histories that consist of a mix of methods and files. To gain a deeper understanding of rule aggregation this section investigates aggregation's behaviour across all four histories. As discussed in subsection 7.2, we differentiate between *theoretical* and *practical* histories, which can be summarized as follows:

Theoretical histories where only parseable artifacts are considered.

Practical histories where all available artifacts are considered.

The next two subsections present results for these two classes of histories.

Theoretical Histories

For the theoretical histories, all artifacts that could not be parsed have been removed. We can therefore focus purely on the effect of granularity without the blurring effect caused when histories share unparseable files. To begin with, Table B.6 shows the percentage of recommendations that were aggregable given histories containing only file level changes, and then separately only method changes. While there is little change for Tarmaq, there is nearly a ten percent increase in applicability for Co-Change when moving from the *theoretical fine-grained* history to the *theoretical coarse-grained* history. At first, this may seem paradoxical, as the number of artifacts increases with finer granularity; however, the opportunities for aggregation may indeed decrease with finer granularity. To see this, consider the visualization shown in Figure B.8. Here, the association rules at the method level have different consequents, while if we lift the rules to the file level, the consequent now becomes shared. Thus, the rules at the method level are non-aggregable (by virtue of having different consequents), while the rules on the file level are aggregable. The converse can not happen however; if two file-level rules are non-aggregable, then all the corresponding method-level rules will be non-aggregable.

Table B.6: Applicability for the theoretical histories.

	theoretical coarse-grained	*theoretical fine-grained*
Co-Change	90%	80%
Tarmaq	14%	13%

While there are more opportunities for aggregation at the file level, both levels provide ample of opportunity; thus we turn to RQ 2, and consider the impact of aggregation using the recommendation based on the non-aggregated rules as a baseline. This involves three steps:

- Starting with the *theoretical coarse-grained* history, for each combination of algorithm, aggregator, and measure, calculate CiMAP (see subsection 8.2)
- Also calculate CiMAP for the *theoretical fine-grained* history.
- Finally, compute the difference between *theoretical coarse-grained* CiMAP and *theoretical fine-grained* CiMAP.

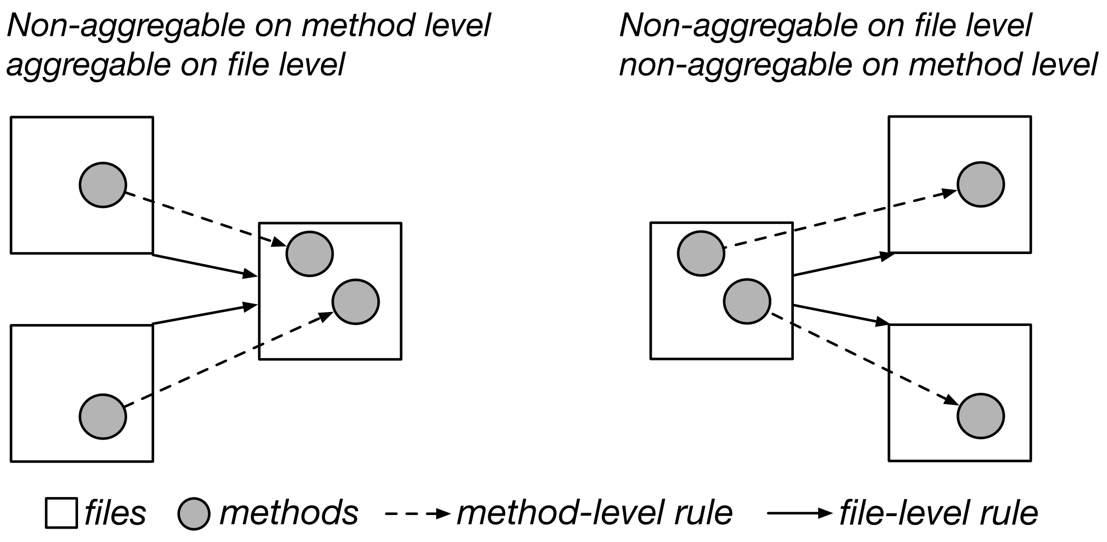

Non-aggregable on method level
aggregable on file level
Non-aggregable on file level
non-aggregable on method level
files
methods
method-level rule
file-level rule

Fig. B.8: Effect of changing granularity level on whether association rules are aggregable or not.

Say that for an interestingness measure, aggregation with the coarse data improved the recommendation by 50% (CiMAP), for that same measure with the fine-grained data, the recommendation improved by 60%. The *delta CiMAP*, given in percentage points, is then $60\% - 50\% = 10$ percentage points.

The results are shown in Figure B.9. In the plot, the *theoretical coarse-grained* history forms the baseline at 0% and the data-set has been split into six sub-plots one for each pairing of algorithm and aggregator. In a sub-plot, each circle gives the result for a single interestingness measure. If a circle lies above the line at $y = 0$, aggregation achieved a higher CiMAP with the *theoretical fine-grained* history than with the *theoretical coarse-grained* history. If the circle lies below the line the converse is true. Note that the scatter-plot nature of Figure B.9 (and Figure B.10) is intentional; the focus here is not on individual interestingness measures but rather on overall trends that emerge when granularity is altered.

Lastly, we performed a Wilcoxon test for each algorithm, measure and aggregator to test for differences in CiMAP of the two granularities. The result are captured by either hollow or filled circles in Figure B.9, where a hollow circle indicates that there was a significant difference. Note that the location of the circle on the y-axis reflects the mean, while the Wilcoxon test compares the distribution of the underlying values, as such it is possible for a circle whose mean is near zero to still show a significant difference.

Overall, Figure B.9 shows that granularity has a stronger effect on the recommendations generated by Co-Change than those generated by Tarmaq: the mean improvement for the *theoretical fine-grained* over the *theoretical coarse-grained* histories is 18.1 percentage points for Co-Change and 7.5 percentage

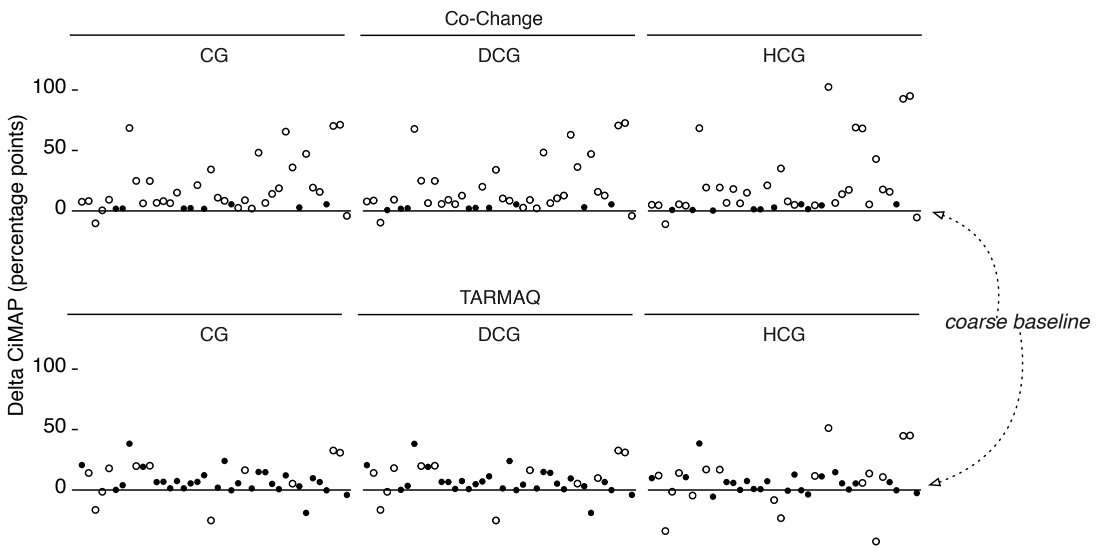


Fig. B.9: Change in the effect of aggregation when moving from *theoretical coarse-grained* to *theoretical fine-grained* histories.

points for Tarmaq. To conclude, the positive effect of aggregation is almost universally more pronounced when using the finer-grained histories.

Practical Histories

In the previous section, only files that could be parsed were considered. However, in practice, transactions often include files that cannot be parsed for various reasons. Example unparseable files include configuration files, binary files, documentation, or simply those of unsupported programming languages. Being able to recommend these types of files is a key advantages of evolutionary coupling when compared to approaches that use static or dynamic source-code analysis. Thus this section repeats the analysis of the previous section and also compares results from the theoretic versus practical histories.

To begin with, Table B.7 presents aggregation's applicability for Co-Change and Tarmaq. Overall the values are slightly higher then those of Table B.6. Considering the differences between the fine and coarse grained histories, they is clearly muted when compared to those seen using the theoretical histories. Thus the data shows the expected damping down of the differences caused by the shared unparseable files.

Finally, we consider precision breakdown shown in Figure B.10, which parallels that shown in Figure B.9. Compared to the theoretical histories, there is less difference between the granularity levels (note the difference in

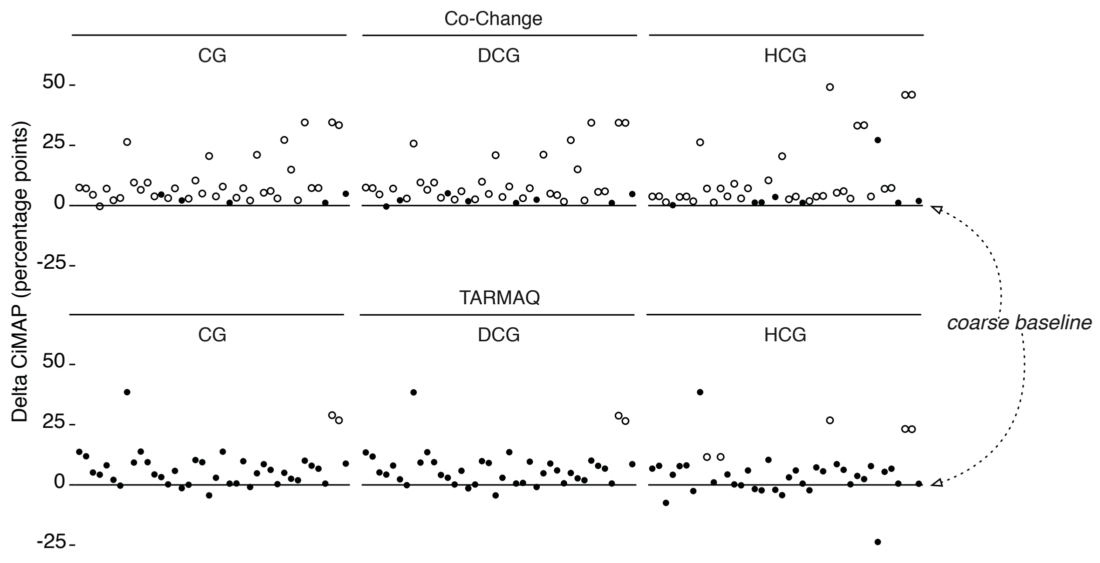


Fig. B.10: Change in the effect of aggregation when moving from *practical coarse-grained* to *practical fine-grained* granular histories.

Table B.7: The percentage of recommendations which were aggregable given histories containing only a mix of method and file changes, or only file changes.

	practical coarse-grained	*practical fine-grained*
Co-Change	89%	84%
Tarmaq	15%	15%

the scale used on the y-axis). Thus while granularity again has a stronger effect on the recommendations generated by Co-Change than those generated by Tarmaq. The effect is not as large: the mean improvement for the *practical fine-grained* over the *practical coarse-grained* histories is 9.5 percentage points for Co-Change and 6.5 percentage points for Tarmaq. The damping effect is also evident in that the Wilcoxon tests find fewer significant differences.

To conclude, aggregation provides less improvement on the practical histories compared to the theoretical histories. Still, the trend for both Co-Change and Tarmaq mirror the theoretical histories: change recommendation based on fine-grained histories benefits more from aggregation than that based on the coarse-grained histories.

8.5 Addendum I: Time Complexity

In our experiment architecture, rule aggregation is implemented as a post-processing step which is not optimal with respect to execution time. In production the generation of hyper rules would be incorporated into an existing rule mining algorithm. Given our non-optimal setup we still found aggregation to have low overhead. In Table B.8 we have summarized the observed execution times for the rule aggregation step, the numbers are given in *milliseconds*.

Table B.8: Execution time for rule aggregation in the context of our experiment.

	algorithm	minimum	q1	median	mean	q3	maximum
1	Co-Change	0.00596	0.56860	2.83200	25.9400	13.1300	8185.0
2	TARMAQ	0.00572	0.03004	0.09465	0.4525	0.3395	499.7

The median execution time for Co-Change was 2.83ms, while the median for Tarmaq was 0.09ms. Aggregation over Co-Change recommendations naturally see higher execution times compared to Tarmaq as the Co-Change algorithm tend to generate more rules. However, only 0.001% of aggregations took more than 1 second.

8.6 Addendum II: Absolute Performance

In earlier sections we have shown that rule aggregation significantly improves the average precision across all interestingness measures. In doing so we have only considered the *relative difference* between aggregated and non-aggregated recommendations. However, it may also be of interest to see the absolute, non-relative, performance of aggregated recommendations. We provide this data in Figure B.11 using the MAP. Here the interestingness measures are ordered based on non-aggregated MAP (e.g., non-aggregated *difference of confidence* achieved the highest MAP for the Co-Change algorithm).

As seen in the figure, the aggregated recommendations always achieve a higher MAP score compared to non-aggregated recommendations[5]. In particular, the highest achieving interestingness measures are further improved through aggregation; referring back to Figure B.4, *difference of confidence* for Co-Change obtained a significant effect size of $r = 0.2$, while for Tarmaq, *gini index* obtained a significant effect size of $r = 0.25$. Perhaps of more interest, a large number of aggregated interestingness measures achieve higher MAP than the single non-aggregated interestingness measure with highest MAP.

9 Threats to validity

Problem Domain used in Evaluation: We evaluated hyper-rules in the context of change recommendations. However, the different interestingness measures studied might not fit well into all problem domains [14]. Still, since we

[5]Exceptions are the *descriptive confirmed confidence* and *example and counterexample rate*, where aggregation was also found to have a non-significant effect in Figure B.4.

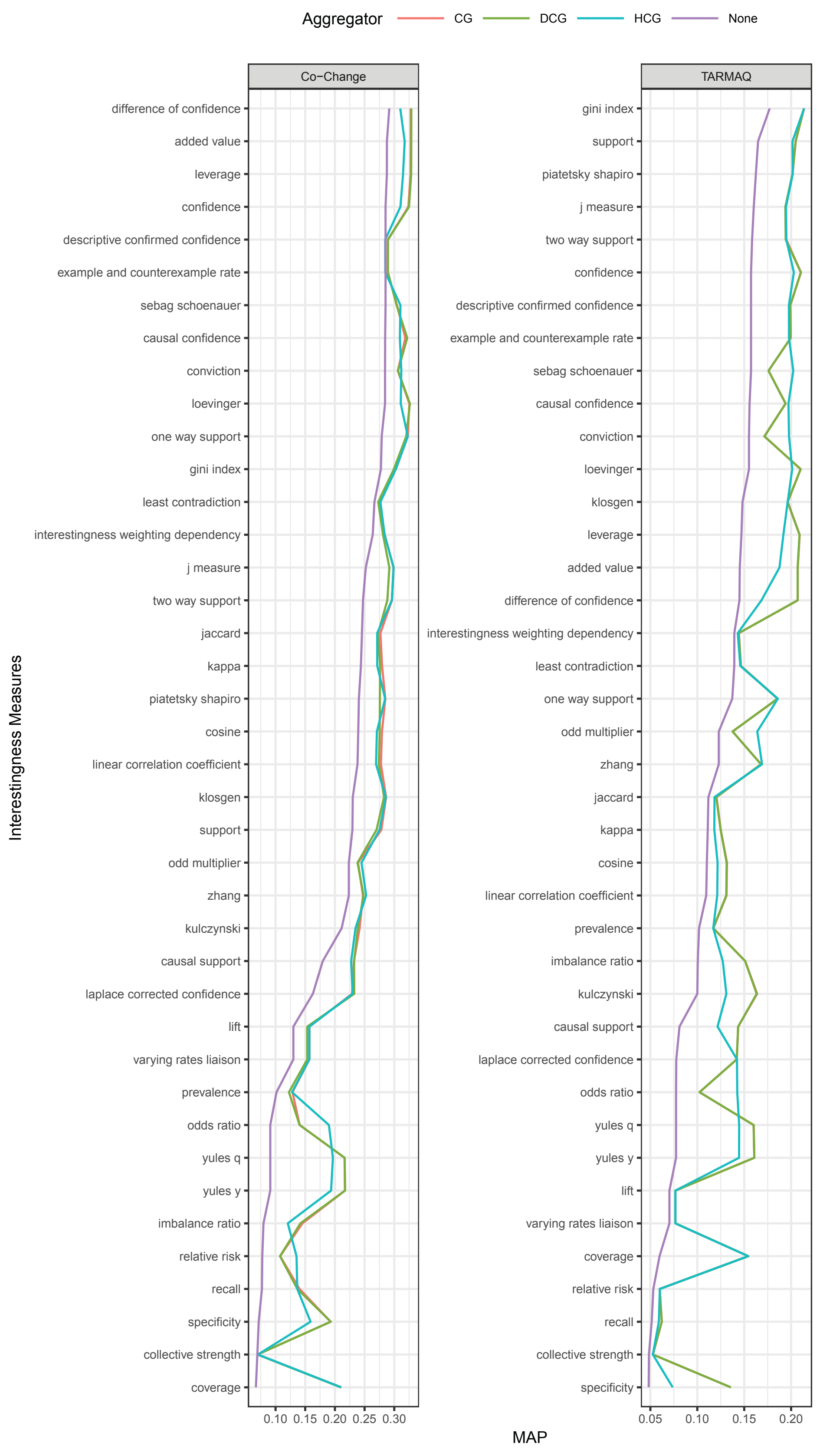


Fig. B.11: MAP of both aggregated and non-aggregated recommendations across different interestingness measures and mining algorithms.

evaluated hyper-rules by looking at the *difference in precision* compared to not using hyper-rules, rather than looking at the actual precision, we believe that the effect of the problem domain is minimized.

Aggregation of only Positive Measures: As discussed in section 4, interestingness measures typically capture either only positive, or both positive and negative correlations between the antecedent and consequent of a rule. In our evaluation however, we only consider positive correlations. While the exact interpretation of negative correlation may differ from measure to measure, the overall interpretation in the context of change recommendations would be: "if this artifact is changed, what artifacts are typically *not* changed?", while we are more interested in the question: "if this artifact is changed, what other artifacts are also typically changed?". Moreover, existing targeted association rule mining algorithms show a clear bias toward mining only positive rules (typically based on artifacts that have changed together in the past). Thus the interestingness measures that have the ability to measure both negative and positive correlations will be heavily skewed towards the positive correlations. Looking forward however, we plan to explore possible beneficial ways of handling the aggregation of both positive and negative correlation.

Implementation: We implemented and thoroughly tested all algorithms, aggregators and interestingness measures studied in this paper in Ruby. However, we can not guarantee the absence of implementation errors which may have affected our evaluation.

Variation in software systems: We evaluated hyper-rules on two industrial systems and 15 large open source systems. These systems vary considerably in size and frequency of transactions (commits), which should provide an accurate picture of the performance of hyper-rules in various contexts. However, despite our careful choice, we are likely not to have captured all variations.

Commits as basis for evolutionary coupling: The evaluation in this paper is grounded in predictions based on the analysis of patterns found in the change histories. The transactions that make up the change histories are however not in any way guaranteed to be "correct" or "complete", in the sense that they represent a coherent unit of work. Non-related files may be present in the transactions, and related files may be missing from the transactions. However, the included software-systems in our evaluation all (except KM) use Git for version control. As Git provides developers with tools for history-rewriting, we do believe that this might cause more coherent transactions.

10 Related Work

We distinguish related work on aggregating association rules, clustering and pruning association rules, and on comparing interestingness measures. Furthermore, we specifically discuss the relation between hyper-rules, and the more familiar techniques of closed and maximal rule mining.

Aggregating Association Rules: To the best of our knowledge, no other work has investigated the aggregation of association rules with the goal of combining the evidence (or interestingness) provided by individual rules. Massoud et al. address the challenge of mining multi-dimensional association rules that aim to combine and relate association rules generated from two or more different sets of transactions [56]. They do not aggregate rules that combine evidence for the same conclusion but aim to create aggregate rules that span the dimensions of all transactions.

Clustering and Pruning Association Rules: Several authors investigate methods to discover the most informative or useful rules in a large collection of mined association rules, for example by clustering rules that convey redundant information, or by pruning *non-interesting* rules. Thus, while our method aims to aggregate rules to combine all existing evidence, this work tries to keep (or only generate) the "most important" rules. Toivonen et al. present *association rule covers* as a method to reduce the number of redundant rules [16]. Their method first groups rules which shared the same consequent, and then filters this set by considering the size of the antecedent in combination with the interestingness measures of the rules. No association rules or interestingness measures are aggregated. Kannan and Bhaskaran build upon Toivonen et al.'s work, and instead consider only rules with high interestingness values when generating clusters of rules sharing the same consequent [57]. In particular, they conclude that extracting clusters from the half of rules with highest interestingness value yields a minimal loss of information. Zaki introduces the *closed* frequent itemset as an alternative association rule mining technique that only generate non-redundant association rules [58]. The number of redundant rules produced by the new approach is dramatically smaller than the rule set from the traditional approach but this is achieved at generation time, i.e., no association rules or interestingness measures are aggregated. Baralis et al. investigate an association rule mining technique that combines *schema constraints* (i.e., rule constraints) and rule taxonomies to filter out redundant rules [59]. As with Zaki's approach, this is achieved at generation time, and no association rules or interestingness measures are aggregated. Liu et al. introduce *direction setting rules* as a method of summarizing the set of rules for a human user [60]. Essentially, direction setting rules are simple rules which capture part of the same relationships also captured in larger rules, i.e., they are more concise.

Selecting and Comparing Interestingness Measures: Tan et al. presents a technique, which given a set of desired properties, can be used to select an appropriate interestingness measure for that context [14]. In this paper, we have not distinguished between interestingness measures in that regard, as such we went for completeness rather than strictly limiting the set of measures to those appropriate for change recommendation. We attempted to control for the potential domain mismatch by only considering the relative improvement between the original and aggregated recommendations. Closely related to Tan et al., Geng and Hamilton survey a range of interestingness measures and discuss properties such as surprisingness and conciseness [15]. In a complementary paper, Vaillant et al. clusters interestingness measures based on empirical performance, rather than theoretical properties [30]. Mcgarry surveys a range of interestingness measures, not only relating to association rules, but patterns for knowledge discovery in general [61]. Of particular relevance here is his discussion of understandability of patterns, for example, the support measure is objectively easier to understand for an end-user compared to the collective strength. When hyper-rules are constructed, we also increase complexity, and perhaps lower understandability of patterns in the change recommendation. As the understanding of patterns is subjective, future work may want to qualitatively study how hyper-rules are interpreted and understood by end-users. Le and Lo evaluate 38 interestingness measures in the context of specification mining, typically; "if File.close has been called, File.open must have been called earlier" [62]. We believe that such temporal, sequence rules, also can benefit from association rule aggregation. In particular, association rule aggregation should be applicable when multiple sequences overlap at at least one point.

Relation to closed and maximal rules: As the number of association rules often grows unwieldy in conventional association rule mining, techniques such as closed and maximal rule mining has been proposed. Both techniques can be used to effectively reduce the number of generated rules, while still being left with the most relevant. A closed rule is a rule for which no superseding rules have the same support [63], while a maximal rule is a rule for which there are no superseding rules that are frequent [64, 65]. In relating closed and maximal rules to hyper-rules, it is most accurate to think about techniques to identify closed and maximal rules as rule mining algorithms, in the same manner that Co-Change and Tarmaq are rule mining algorithms. In this paper we have considered Co-Change and Tarmaq as they are targeted association rule mining algorithms, which fit the problem domain of change recommendation. In doing so, we explored hyper-rules which aggregate rules with the same consequent, as previously stated, hyper-rules could also be formed based on other selection criteria. Association rule aggregation is agnostic to the origin of the generated rules, it is up to the user to define rule clusters which could benefit from aggregation. We therefore strongly believe that there is potential for association rule aggregation *over closed or maximal rules*. Not only could this further reduce the number of rules, but also improve relevance of the resulting rule set.

11 Concluding Remarks

Association rules capture knowledge found in the relationships between artifacts. In this paper we present a technique for *rule aggregation*. The resulting hyper rules combine knowledge from sets of conventional association rules in a beneficial way. This paper extends and complements our initial work on this topic, which introduced the notion of hyper rules [17]; however, this paper is self contained and makes the following contributions: **(1)** We identify an opportunity, missed by traditional recommendation systems, to increase accuracy using the evidence of multiple applicable rules in support of a particular conclusion. **(2)** We provide a theoretical foundation for rule aggregation through the concept of hyper rules. **(3)** We present three aggregation strategies for forming hyper rules, where two are adapted from Information Retrieval and one is introduced in this paper. **(4)** We provide formal proofs that all studied aggregators satisfy the set of desirable properties given in Average Precision 4 for non-negative values. **(5)** We perform a large empirical study where hyper rules are evaluated in the context of change recommendation. We include systems from our two industry partners, Cisco and Kongsberg Maritime, as well as 15 open source systems. Furthermore, for each system, four different histories are studied. The histories vary in terms of their *granularity* (file level versus method level) and *parsability*. Our findings are as follows: **(result.a)** We find that, depending on the underlying history and rule mining algorithm, between approximately 13% and 90% of the generated change recommendations are candidates for association rule aggregation. **(result.b)** Of the 40 studied interestingness measures, we find that rule aggregation significantly improves the precision of change recommendation for all but two interestingness measures when used with Co-Change and for all measures when used with Tarmaq. **(result.c)** We find that aggregation performance varies across software-systems, but the effect is consistently positive. **(result.d)** In our study of history granularity, we find that typically, finer grained histories benefit more from rule aggregation than the coarse histories. Furthermore, histories where all artifacts are parseable, also see more benefit from rule aggregation than histories that contain a mix of parseable and unparseable artifacts. We conjecture that this is the case because those histories only contain source-code artifacts, which are more likely to show relevant co-change patterns.

Directions for Future Work: In the future we would like to address the following. **(1)** Generally, different association rules may be generated from the same transactions. When aggregating such rules it might be beneficial to account for these overlaps. **(2)** While we found that the studied aggregators did not benefit from negative correlations, we plan to explore alternative aggregation possibilities in the future. **(3)** We also plan to consider how the tie-breaking mechanism used by HCG can be used to create more effective variants of other aggregation functions. **(4)** The experiments studied the effect of aggregation using *all* the rules generated by Co-Change and Tarmaq. A natural extension would be to explore interestingness constraints. For example, the use of a minimum support value to produce *frequent hyper rules*, where only frequent rules are aggregated. **(5)** We would like to investigate the effect of rule aggregation on other transaction definitions, e.g., sliding windows. **(6)** We also plan to investigate the behavior of hyper rules when using other association rule mining algorithms from the change-recommendation domain. **(7)** This will be complimented by the exploration of uses for hyper rules in other domains. **(8)** Finally, with regards to weighting hyper rules, rather than calculating aggregated measures, we conjecture that new interestingness measures can be defined directly in terms of the hyper rules.

Acknowledgements:
This work is supported by the Research Council of Norway through the EvolveIT project (#221751/F20) and the Certus SFI (#203461/030). Dr. Binkley is supported by NSF grant IIA-1360707 and a J. William Fulbright award.

Appendix: Proofs

Within this section we formally prove that DCG and HCG satisfies the properties of Average Precision 4. As expressed earlier, we have limited our study of aggregation functions to positive values. We leave out the proof for CG as it is simply the algebraic sum and therefore naturally satisfies all properties of Average Precision 4.

11.1 Proof for Discounted Cumulative Gain

Theorem 1. *DCG (Definition 6) satisfies the properties in Definition 4 for non-negative values of an interestingness measure M.*

Proof. Let M be an interestingness measure, R be a set of rules with non-negative interestingness values, and $V = [v_1, \ldots, v_n]$ be an ordered list of interestingness values of rules in R for measure M, such that $\forall\, i < j.\ v_i \geq v_j$. Let r be an arbitrary rule in R, and R' be equal to $R \setminus \{r\}$. Then, there exists a $v_k \in V$ such that $M(r) = v_k$. We define $U = [u_1, \ldots, u_{n-1}]$ as the ordered list of interestingness values of rules in R' for measure M. U is equal to V

except that v_k is removed from it, and we have $\forall\ i < j.\ u_i \geq u_j$. Then:

$$\forall 1 \leq i < k \ .\ v_i = u_i \tag{B.6}$$

$$\forall k < i \leq n \ .\ v_i = u_{i-1} \tag{B.7}$$

The first property in Average Precision 4, which concerns Rs of size one, is trivial. To prove that the second property in Average Precision 4 holds for DCG, we compute the difference between the DCG of R and the DCG of R' and show that for $v_k > 0$, this difference is positive, and for $v_k = 0$, this difference is equal to zero. Let $DCG(R, M)$ and $DCG(R', M)$ denote the DCGs of R and R' for the interestingness measure M, respectively.

$$\begin{aligned}
DCG(R,M) - DCG(R',M) &= \sum_{i=1}^{n} \frac{v_i}{log_2(i+1)} - \sum_{i=1}^{n-1} \frac{u_i}{log_2(i+1)} \\
&= \sum_{i=1}^{k-1} \frac{v_i - v_i}{log_2(i+1)} \\
&+ \sum_{i=k}^{n} \frac{v_i}{log_2(i+1)} - \sum_{i=k}^{n-1} \frac{u_i}{log_2(i+1)} \\
&\overset{Eq\ B.6,B.7}{=} \sum_{i=k}^{n-1} \frac{v_i - v_{i+1}}{log_2(i+1)} + \frac{v_n}{log_2(n+1)} \qquad \text{(B.8)}
\end{aligned}$$

Note that both terms in the last line are always non-negative. Now, we consider two cases based on the value of v_k:

$\boldsymbol{v_k > 0}$ – Since $v_k > 0$, the two terms in Eq. B.8 cannot be zero simultaneously. Because this requires $v_n = 0$, and at the same time $\forall i \geq k, v_i = v_{i+1}$. The latter implies $v_n = v_k > 0$, which contradicts with the former. Therefore, in this case, there is always at least one positive term in Eq. B.8. Thus:

$$DCG(R,M) - DCG(R',M) > 0 \tag{B.9}$$

$\square$

This proves that the second property in Definition 4 holds when $M(r) = v_k$ is positive.

$\boldsymbol{v_k = 0}$ – In this case $DCG(R, M) - DCG(R', M) = 0$. This follows from the original assumption that the rules in V are ordered according to their absolute values. Therefore, in Eq. B.8 all v_is are equal to zero.

11.2 Proof for Hyper Cumulative Gain

We now prove that HCG satisfies the monotonicity properties of Average Precision 4 for non-negative values of an interestingness measure M. We start by introducing a new operator, and two lemmas that are used in the proof.

Definition 13 (Correlative sum). *Let a_1 and a_2 be real numbers. For any nonzero real number b, we define the operator S_b as*

$$a_1\ S_b\ a_2 = a_1 + \frac{b - a_1}{b} \cdot a_2.$$

Lemma 1 (Properties of correlative sum). *For any nonzero real number b, the correlative sum S_b is commutative, and associative.*

Proof. S_b is commutative:

$$\begin{aligned} a_1\ S_b\ a_2 &= a_1 + \frac{b - a_1}{b} \cdot a_2 \\ &= \frac{a_1 b + a_2 b - a_1 a_2}{b} \\ &= a_2 + \frac{b - a_2}{b} \cdot a_1 \\ &= a_2\ S_b\ a_1 \end{aligned}$$

S_b is associative:

$$\begin{aligned} (a_1\ S_b\ a_2)\ S_b\ a_3 &= a_1 + \frac{b - a_1}{b} \cdot a_2 + \frac{b - (a_1 + \frac{b - a_1}{b} \cdot a_2)}{b} \cdot a_3 \\ &= \frac{a_1 b + a_2 b - a_1 a_2}{b} + \frac{b - (\frac{a_1 b + a_2 b - a_1 a_2}{b})}{b} \cdot a_3 \\ &= \frac{a_1 b + a_2 b - a_1 a_2}{b} + \frac{b^2 - (a_1 b + a_2 b - a_1 a_2)}{b^2} \cdot a_3 \\ &= \frac{a_1 b^2 + a_2 b^2 - a_1 a_2 b + a_3 b^2 - a_1 a_3 b - a_2 a_3 b + a_1 a_2 a_3}{b^2} \\ &= a_1 + \frac{b - a_1}{b} \cdot \frac{a_2 b + a_3 b - a_2 a_3}{b} \\ &= a_1 + \frac{b - a_1}{b} \cdot (a_2 + \frac{b - a_2}{b} \cdot a_3) \\ &= a_1\ S_b\ (a_2\ S_b\ a_3) \end{aligned}$$

An important implication of this lemma is that S_b can be applied to a sequence of numbers independent from the ordering of the elements in the sequence.

Lemma 2. *For any nonzero real number b, let $L = \{l_1, l_2, ..., l_n\}$ be a sequence of real numbers. Let $S_b(L)$ denote $l_1\ S_b\ l_2 \cdots\ l_{n-1}\ S_b\ l_n$. Then*

$$S_b(L) = l_1 + \sum_{i=2}^{n} \left(l_i \cdot \prod_{j=1}^{i-1}(1 - \frac{l_j}{b})\right) \tag{B.10}$$

and for any given real number l, we have:

$$S_b(L \cup \{l\}) = S_b(L) + l \cdot \prod_{l_j \in L}(1 - \frac{l_j}{b}). \tag{B.11}$$

Proof. The proof for both parts is straightforward after expanding the polynomials.

An implication of Equation B.11 is that, for any sequence of real numbers $L = \{l_1, l_2, ..., l_n\}$, and any arbitrary real number l in L, we have:

$$S_b(L) - S_b(L \setminus \{l\}) \ = \ l \cdot \prod_{l_j \in L \setminus \{l\}} (1 - \frac{l_j}{b}) \tag{B.12}$$

Theorem 2. *HCG (Correlative sum 9) satisfies the properties of* Correlative sum 4 *for non-negative values of an interestingness measure M.*

Proof. Let M be a normalized interestingness measure with upper bound $b \in \mathbb{R} \setminus \{0\}$, $R = \{r_1, \ldots, r_n\}$ be a set of rules, and $L_M = \langle M(r_1), \cdots M(r_n)\rangle$ be the sequence of non-negative interestingness values for the rules in R. HCG of $\mathcal{H}(R)$ for M defined in Definition 9 can be rewritten as follows:

$$HCG(\mathcal{H}(R), M) = \Big(S_b(L_M),\ m\Big) \tag{B.13}$$

where m is given by

$$m = |\{r \in R \mid M(r) > 0\}|$$

The first property in Correlative sum 4, which concerns Rs of size one, is again trivial. Let r be an arbitrary rule in R, and l denote $M(r)$. To prove that the second property in Correlative sum 4 holds for HCG, we compare the HCGs of R and $R \setminus \{r\}$, and consider three cases based on the value of $M(r)$:

$\boldsymbol{M(r) > 0}$ – In this case, we need to show that the HCG of R is greater than the HCG of $R \setminus \{r\}$.

$$\begin{aligned} & HCG(\mathcal{H}(R), M) > HCG(\mathcal{H}(R \setminus \{r\}), M) && \text{(B.14)} \\ & \overset{Eq\ B.13}{\equiv} \Big(S_b(L_M),\ m\Big) > \Big(S_b(L_M \setminus \{l\}),\ m-1\Big) \\ & \overset{Def\ 10}{\equiv} S_b(L_M) \geq S_b(L_M \setminus \{l\}) && \text{(B.15)} \end{aligned}$$

To show that inequality B.15 holds we show that $S_b(L_M) - S_b(L_M \setminus \{l\})$ is non-negative, which, according to Equation B.11, is equivalent to showing that

$$l \cdot \prod_{l_j \in L_M \setminus \{l\}} (1 - \frac{l_j}{b}) \geq 0$$

This inequality holds because $l = M(r) > 0$, and for all $l_j \in L_M \setminus \{l\}$ the term $1 - \frac{l_j}{b}$ is non-negative (because l_j is at most b). Since $m > m - 1$, inequality B.15 holds, completing the case for $M(r) > 0$.

$\boldsymbol{M(r) = 0}$ – In this case, we have to show that the HCGs of R and $R \setminus \{r\}$ are equal.

$$\begin{aligned} HCG(\mathcal{H}(R), M) &= HCG(\mathcal{H}(R \setminus \{r\}), M) \\ &\overset{Eq\ B.13}{\equiv} \Big(S_b(L_M),\ m\Big) = \Big(S_b(L_M \setminus \{l\}),\ m\Big) \end{aligned} \tag{B.16}$$

To do so, we have to show that $S_b(L_M) = S_b(L_M \setminus \{l\})$. This is proven by forming $S_b(L) - S_b(L_M \setminus \{l\})$, and replacing l with zero in the right-hand side of Equation B.11. Therefore, completing the proof.

So far, we have proven that HCG satisfies the properties in Correlative sum 4 for interestingness measures that have a finite upper bound. If the upper bound of an interestingness measures is infinity, then like CG, HCG becomes the sum of interestingness values of all rules in R. Therefore, it satisfies all the properties in Correlative sum 4.

References

[1] G. Canfora and L. Cerulo, “Impact Analysis by Mining Software and Change Request Repositories,” in *International Software Metrics Symposium (METRICS)*. IEEE, 2005, pp. 29–37. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1509307http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1509307

[2] M.-A. Jashki, R. Zafarani, and E. Bagheri, “Towards a more efficient static software change impact analysis method,” in *ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE)*. ACM, 2008, pp. 84–90. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1512475.1512493

[3] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, “Chianti: a tool for change impact analysis of java programs,” in *ACM*

SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA), 2004, pp. 432–448. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1035292.1029012

[4] M. B. Zanjani, G. Swartzendruber, and H. Kagdi, "Impact analysis of change requests on source code based on interaction and commit histories," in *International Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 162–171. [Online]. Available: http://dl.acm.org/citation.cfm?id=2597096http://dx.doi.org/10.1145/2597073.2597096

[5] S. Bohner and R. Arnold, *Software Change Impact Analysis*. CA, USA: IEEE, 1996.

[6] A. R. Yazdanshenas and L. Moonen, "Crossing the boundaries while analyzing heterogeneous component-based software systems," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011, pp. 193–202. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2011.6080786http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6080786

[7] A. Podgurski and L. Clarke, "A formal model of program dependences and its implications for software testing, debugging, and maintenance," *IEEE Transactions on Software Engineering*, vol. 16, no. 9, pp. 965–979, 1990. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=58784

[8] S. H. Yong and S. Horwitz, "Reducing the Overhead of Dynamic Analysis," *Electronic Notes in Theoretical Computer Science*, vol. 70, no. 4, pp. 158–178, dec 2002. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1571066104805838

[9] C. Bird, T. Menzies, and T. Zimmermann, "Past, Present, and Future of Analyzing Software Data," in *The Art and Science of Analyzing Software Data*, 2015, pp. 1–13. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978012411519400001X

[10] S. Eick, T. L. Graves, A. Karr, J. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 1–12, 2001. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=895984

[11] R. Robbes, D. Pollet, and M. Lanza, "Logical Coupling Based on Fine-Grained Change Information," in *Working Conference on Reverse Engineering (WCRE)*. IEEE, 2008, pp. 42–46. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4656392

[12] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD International Conference on Management of Data*. ACM, 1993, pp. 207–216. [Online]. Available: http://portal.acm.org/citation.cfm?doid=170035.170072

[13] M. Kamber and R. Shinghal, "Evaluating the Interestingness of Characteristic Rules," in *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1996, pp. 263–266.

[14] P.-N. Tan, V. Kumar, and J. Srivastava, "Selecting the right objective measure for association analysis," *Information Systems*, vol. 29, no. 4, pp. 293–313, jun 2004. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0306437903000723

[15] L. Geng and H. J. Hamilton, "Interestingness measures for data mining," *ACM Computing Surveys*, vol. 38, no. 3, sep 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1132960.1132963

[16] H. Toivonen, M. Klemettinen, P. Ronkainen, K. Hätönen, and H. Mannila, "Pruning and Grouping Discovered Association Rules," in *Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases*, Heraklion, Crete, Greece, 1995, pp. 47–52.

[17] T. Rolfsnes, L. Moonen, S. Di Alesio, R. Behjati, and D. W. Binkley, "Improving change recommendation using aggregated association rules," in *International Conference on Mining Software Repositories (MSR)*. ACM, 2016, pp. 73–84. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2901739.2901756

[18] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 1998, pp. 190–198. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=738508

[19] R. Srikant, Q. Vu, and R. Agrawal, "Mining Association Rules with Item Constraints," in *International Conference on Knowledge Discovery and Data Mining (KDD)*. AASI, 1997, pp. 67–73.

[20] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1463228

[21] A. T. T. Ying, G. Murphy, R. T. Ng, and M. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, 2004.

[Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1324645

[22] T. Rolfsnes, S. Di Alesio, R. Behjati, L. Moonen, and D. W. Binkley, "Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis," in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, mar 2016, pp. 201–212. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7476643

[23] T. Ball, J. Kim, and H. P. Siy, "If your version control system could talk," in *ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering*, 1997. [Online]. Available: http://csalpha.ist.unomaha.edu/{~}hsiy/research/visual.pdf

[24] D. Beyer and A. Noack, "Clustering Software Artifacts Based on Frequent Common Changes," in *International Workshop on Program Comprehension (IWPC)*. IEEE, 2005, pp. 259–268. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1421041

[25] A. E. Hassan and R. Holt, "Predicting change propagation in software systems," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2004, pp. 284–293. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1357812

[26] Y. Kodratoff, "Comparing Machine Learning and Knowledge Discovery in DataBases: An Application to Knowledge Discovery in Texts," in *Machine Learning and Its Applications, LNAI 2049*. Springer, 2001, ch. 1, pp. 1–21. [Online]. Available: http://link.springer.com/10.1007/3-540-44673-7{_}1

[27] C. C. Aggarwal and P. S. Yu, "A new framework for itemset generation," in *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, no. 2. ACM, 1998, pp. 18–24. [Online]. Available: http://portal.acm.org/citation.cfm?doid=275487.275490

[28] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket data," in *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, vol. 26, no. 2. ACM, jun 1997, pp. 255–264. [Online]. Available: http://portal.acm.org/citation.cfm?doid=253262.253325http://portal.acm.org/citation.cfm?doid=253260.253325

[29] H. Hofmann and A. Wilhelm, "Visual Comparison of Association Rules," *Computational Statistics*, vol. 16, no. 3, pp. 399–415, sep 2001. [Online]. Available: http://link.springer.com/10.1007/s001800100075

[30] B. Vaillant, P. Lenca, and S. Lallich, "A Clustering of Interestingness Measures," in *Lecture Notes in Artificial Intelligence (LNAI)*, 2004, vol. 3245, pp. 290–297. [Online]. Available: http://link.springer.com/10.1007/978-3-540-30214-8{_}23

[31] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*, 1984, vol. 19.

[32] T. Wu, Y. Chen, and J. Han, "Re-examination of interestingness measures in pattern mining: a unified framework," *Data Mining and Knowledge Discovery*, vol. 21, no. 3, pp. 371–397, nov 2010. [Online]. Available: http://link.springer.com/10.1007/s10618-009-0161-2

[33] B. Gray and M. E. Orlowska, "CCAIIA: Clustering categorical attributes into interesting association rules," in *Lecture Notes in Computer Science (LNCS)*, 1998, vol. 1394, pp. 132–143. [Online]. Available: http://link.springer.com/10.1007/3-540-64383-4{_}12

[34] P. Smyth and R. Goodman, "An information theoretic approach to rule induction from databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 4, no. 4, pp. 301–316, 1992. [Online]. Available: http://ieeexplore.ieee.org/document/149926/

[35] C. J. Van Rijsbergen, *Information Retrieval*. Butterworth-Heinemann, 1979.

[36] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, apr 1960. [Online]. Available: http://epm.sagepub.com/cgi/doi/10.1177/001316446002000104

[37] W. Klösgen, "Problems for knowledge discovery in databases and their treatment in the statistics interpreter explora," *International Journal of Intelligent Systems*, vol. 7, no. 7, pp. 649–673, 1992. [Online]. Available: http://doi.wiley.com/10.1002/int.4550070707

[38] S. Kulczyński, *Die Pflanzenassoziationen der Pieninen*. Imprimerie de l'Université, 1928.

[39] I. J. Good, *The estimation of probabilities: An essay on modern Bayesian methods*. MIT Press, 1966. [Online]. Available: https://mitpress.mit.edu/books/estimation-probabilities

[40] J. Azé and Y. Kodratoff, "Evaluation de la résistance au bruit de quelques mesures d'extraction de règles d'association." in *Extraction et Gestion des Connaissances (EGC)*, vol. 1, no. 4. Hermes Science Publications, 2002, pp. 143–154. [Online]. Available: http://dblp.uni-trier.de/rec/bib/conf/f-egc/AzeK02

[41] G. Piatetsky-Shapiro, “Discovery, analysis, and presentation of strong rules,” *Knowledge discovery in databases*, pp. 229—-238, 1991.

[42] K. Pearson, “Mathematical Contributions to the Theory of Evolution. III. Regression, Heredity, and Panmixia,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 187, pp. 253–318, jan 1896. [Online]. Available: http://rsta.royalsocietypublishing.org/cgi/doi/10.1098/rsta.1896.0007

[43] J. Loevinger, “A systematic approach to the construction and evaluation of tests of ability.” *Psychological Monographs*, vol. 61, no. 4, pp. i–49, 1947. [Online]. Available: http://doi.apa.org/getdoi.cfm?doi=10.1037/h0093565

[44] F. Mosteller, “Association and Estimation in Contingency Tables,” *Journal of the American Statistical Association*, vol. 63, no. 321, pp. 1–28, mar 1968. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01621459.1968.11009219

[45] Y. Y. Yao and N. Zhong, “An Analysis of Quantitative Measures Associated with Rules,” in *Methodologies for knowledge discovery and data mining (LNCS 1574)*. Springer, 1999, pp. 479–488. [Online]. Available: http://www.springerlink.com/index/e62m8gpjeyk1nbeq.pdfhttp://link.springer.com/10.1007/3-540-48912-6{_}64

[46] M. Sebag and M. Schoenauer, “Generation of rules with certainty and confidence factors from incomplete and incoherent learning bases,” in *Proceedings of the European Knowledge Acquisition Workshop (EKAW)*, 1988, p. 28.

[47] J.-M. Bernard and C. Charron, “Bayesian implicative analysis, a method for the study of oriented dependencies,” *Mathématiques, Informatique et Sciences Humaines*, vol. 135, pp. 5–18, 1996.

[48] G. U. Yule, “On the Association of Attributes in Statistics,” *Philosophical Transactions of the Royal Society of London*, vol. 194, pp. 257–319, 1900.

[49] ——, “On the methods of measuring association between two attributes,” *Journal of the Royal Statistical Society*, vol. LXXV, pp. 579–652, 1912.

[50] T. Zhang, “Association Rules,” in *Knowledge Discovery and Data Mining. Current Issues and New Applications*, 2000, no. c, pp. 245–256. [Online]. Available: http://link.springer.com/10.1007/3-540-45571-X{_}31

[51] K. Järvelin and J. Kekäläinen, “Cumulated gain-based evaluation of IR techniques,” *ACM Transactions on Information Systems*, vol. 20, no. 4, pp.

422–446, oct 2002. [Online]. Available: http://portal.acm.org/citation.cfm?doid=582415.582418

[52] L. Moonen, S. Di Alesio, T. Rolfsnes, and D. W. Binkley, "Exploring the Effects of History Length and Age on Mining Software Change Impact," in *IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, sep 2016, pp. 207–216.

[53] M. L. Collard, M. J. Decker, and J. I. Maletic, "srcML: An Infrastructure for the Exploration, Analysis, and Manipulation of Source Code: A Tool Demonstration," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, sep 2013, pp. 516–519. [Online]. Available: http://ieeexplore.ieee.org/document/6676946/

[54] R. Rosenthal, *Meta-analytic procedures for social research*. SAGE, 1991.

[55] J. Cohen, "A power primer." *Psychological Bulletin*, vol. 112, no. 1, pp. 155–159, apr 1992. [Online]. Available: http://www.nature.com/doifinder/10.1038/141613a0http://doi.apa.org/getdoi.cfm?doi=10.1037/0033-2909.112.1.155

[56] R. B. Messaoud, S. L. Rabaséda, O. Boussaid, and R. Missaoui, "Enhanced mining of association rules from data cubes," in *International Workshop on Data warehousing and OLAP (DOLAP)*. ACM, 2006, p. 11. [Online]. Available: http://dl.acm.org/citation.cfm?id=1183517http://portal.acm.org/citation.cfm?doid=1183512.1183517

[57] S. Kannan and R. Bhaskaran, "Association Rule Pruning based on Interestingness Measures with Clustering," *Journal of Computer Science*, vol. 6, no. 1, pp. 35–43, dec 2009.

[58] M. J. Zaki, "Generating non-redundant association rules," in *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2000, pp. 34–43. [Online]. Available: http://portal.acm.org/citation.cfm?doid=347090.347101

[59] E. Baralis, L. Cagliero, T. Cerquitelli, and P. Garza, "Generalized association rule mining with constraints," *Information Sciences*, vol. 194, pp. 68–84, 2012. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0020025511002659

[60] B. Liu, W. Hsu, and Y. Ma, "Pruning and summarizing the discovered associations," in *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 1999, pp. 125–134. [Online]. Available: http://portal.acm.org/citation.cfm?doid=312129.312216

[61] K. McGarry, "A survey of interestingness measures for knowledge discovery," *The Knowledge Engineering Review*, vol. 20, no. 01, p. 39, 2005.

[62] T.-d. B. Le and D. Lo, "Beyond support and confidence: Exploring interestingness measures for rule-based specification mining," in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 331–340. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7081843

[63] M. J. Zaki and C.-J. Hsiao, "CHARM: An efficient algorithm for closed association rule mining," *2nd SIAM International Conference on Data Mining*, pp. 457–473, 1999. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.2956{&}rep=rep1{&}type=pdf

[64] R. J. Bayardo, "Efficiently mining long patterns from databases," *ACM SIGMOD Record*, vol. 27, no. 2, pp. 85–93, jun 1998. [Online]. Available: http://cs.sungshin.ac.kr/{~}de/Seminar/ps{_}files/sigmod98{_}max.pdfhttp://portal.acm.org/citation.cfm?doid=276305.276313

[65] D.-I. Lin and Z. M. Kedem, "Pincer-search: A new algorithm for discovering the maximum frequent set," 1998, pp. 103–119. [Online]. Available: http://link.springer.com/10.1007/BFb0100980

Paper C

Practical Guidelines for Change Recommendation using Association Rule Mining

Leon Moonen, Stefano Di Alesio, Dave W. Binkley and Thomas Rolfsnes

Accepted for publication in the main research track of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), September 3-7, 2016.

© 2016 IEEE
The layout has been revised.

Abstract

Association rule mining is an unsupervised learning technique that infers relationships among items in a data set. This technique has been successfully used to analyze a system's change history and uncover evolutionary coupling *between system artifacts. Evolutionary coupling can, in turn, be used to* recommend *artifacts that are potentially* affected *by a given* set of changes *to the system. In general, the quality of such recommendations is affected by (1) the values selected for various parameters of the mining algorithm, (2) characteristics of the set of changes used to derive a recommendation, and (3) characteristics of the system's change history for which recommendations are generated.*

In this paper, we empirically investigate the extent to which certain choices for these factors affect change recommendation. Specifically, we conduct a series of systematic experiments on the change histories of two large industrial systems and eight large open source systems, in which we control the size of the change set for which to derive a recommendation, the measure used to assess the strength of the evolutionary coupling, and the maximum size of historical changes taken into account when inferring these couplings. We use the results from our study to derive a number of practical guidelines for applying association rule mining for change recommendation.

1 Introduction

A well-know effect of the continued evolution of a software system is the increasing disorder or entropy in the system: as a result of repeated changes, the number and complexity of dependencies between parts of the code grows, making it increasingly difficult for developers to foresee and reason about the effects of changes they make to a system.

Automated *change impact analysis* techniques [1–4] aim to support a developer during system evolution by identifying the artifacts (e.g., files, methods, or classes) affected by a given change. Traditionally, these change impact analysis techniques are based on static or dynamic dependency analysis [5] (for example, by identifying the methods that call a changed method). More recently, promising alternative techniques have been proposed that identify dependencies by means of *evolutionary coupling*. These alternative approaches avoid certain limitations in existing techniques. For example, static and dynamic dependency analysis are generally language-specific, making them unsuitable for the analysis of heterogeneous software systems [6]. In addition, they can involve considerable overhead (e.g., dynamic analysis' need for code-instrumentation), and tend to over-approximate the impact of a change [7].

Evolutionary couplings differ from the ones found through static and dynamic dependency analysis, in that they are based on *how* the software was changed over time. In essence, evolutionary coupling aims to build on the developer's inherent knowledge of the dependencies in the system, which can manifest themselves by means of commit-comments, bug-reports, context-switches in an IDE, etc. In this paper, we consider *co-change* as the basis for uncovering evolutionary coupling. Co-change information can, for example, be extracted from a project's version control system [8], from its issue tracking system, or by instrumenting the development environment [9].

The most frequently used method for mining evolutionary coupling from co-change data is *association rule mining* (also called *association rule learning*) [10]. Various variants on the approach have been described in the software engineering literature [11–14]. All of these approaches have in common that the technique is tuned with a number of parameters. While studying the literature, we found that there is little to no practical guidance on the tuning of these parameters, nor has there been a systematic evaluation of their impact on change recommendation quality. Moreover, we conjecture that, in addition to parameters of the mining algorithm, the recommendation quality is also affected by characteristics of the change set that is used to derive the recommendation, and by characteristics of the system's change history from which the recommendation is generated.

In this paper, we empirically investigate the extent to which these factors affect change recommendation. Specifically, we conduct a series of systematic experiments on the change histories of two large industrial systems and eight large open source systems. In these experiments we control the size of change set used to derive recommendations, the measure used to assess the strength of the evolutionary coupling, and the maximum size of historical changes taken into account when inferring association rules. We use the results from our study to derive practical guidelines for applying association rule mining to construct software change recommendations.

Contributions: This paper presents three key contributions: (1) we investigate a previously unexplored area of tuning association rule mining parameters for software change recommendation; (2) we evaluate how recommendation quality is impacted by both characteristics of the change set used to derive a recommendation, and characteristics of change history used for learning; (3) we derive practical guidelines for improving the application of association rule mining in the derivation of software change recommendations.

Overview: The remainder of this paper is organized as follows: section 2 provides background on targeted association rule mining. section 3 describes limitations of classical approaches. section 4 describes the setup of our empirical investigation whose results are presented in section 5. Finally, section 6 presents the related work and then section 7 provides some concluding remarks.

2 Association Rule Mining

Agrawal et al. introduced the concept of *association rule mining* as a discipline aimed at inferring relations between *entities* of a dataset [10]. *Association rules* are implications of the form $A \rightarrow B$, where A is referred to as the *antecedent*, B as the *consequent*, and A and B are disjoint sets. For example, consider the classic application of analyzing shopping cart data: if multiple transactions include bread and butter then a potential association rule is *bread* $\rightarrow$ *butter*. This rule can be read as *"if you buy bread, then you are also likely to buy butter."*

In the context of mining evolutionary coupling from co-change information, the entities involved are the files of the system[1] and a collection of transactions, denoted $\mathcal{T}$ (e.g., a history of commits to the version control system). A transaction $T \in \mathcal{T}$ is the set of files that were either changed or added while addressing a given bug or feature request, hence creating a *logical dependence* between the files [15].

As originally defined [10], association rule mining generates rules that express patterns in a complete data set. However, some applications can exploit a more focused set of rules. *Targeted association rule mining* [16] focuses the generation of rules by applying a constraint. One example constraint specifies that the antecedent of all mined rules belongs to a particular set of files, which effectively reduces the number of rules that need to be created. This reduction can drastically improve rule generation time [16].

When performing change impact analysis, rule constraints are based on a *change set*, for example, the set of files that were modified since the last commit. This set is also referred to as a *query* for which to search for potential impact. In the constrained case, only rules with at least one changed entity in the antecedent are created. The output of change impact analysis is the set of files from the system historically changed alongside the elements of the change set. For example, given the change set $\{a, b, c\}$, change impact analysis would uncover files that were changed when a, b, and c were changed. The resulting impacted files are those found in the rule consequents. These files can be ranked based on the rule's *interestingness measure*.

To our knowledge, only a few targeted association rule mining algorithms have been considered in the context of change impact analysis: Zimmerman et al. [11], Ying et al. [12], Kagdi et al. [13], and our previous work [14]. In contrast, simpler *co-change* algorithms have been well studied in a variety of contexts [15, 17–19]. Existing targeted association rule mining algorithms and the co-change algorithms differ in terms of which subsets of the change set are allowed in the antecedent of generated rules. Consider, for example, the

[1] Other granularities are possible, and our choice of file-level changes is without loss of generality as our algorithms are granularity agnostic: if fine-grained co-change data is available, the algorithms will relate methods or variables just as well as files.

subsets of the change set $C = \{a, b, c, d\}$:

$$\begin{aligned} \mathit{powerset}(C) = \{&\{\}, \\ &\{a\}, \{b\}, \{c\}, \{d\}, && (1) \\ &\{a,b\}, \{a,c\}, \{a,d\}, \{b,c\}, \{b,d\}, \{c,d\}, && (2) \\ &\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}, && (3) \\ &\{a,b,c,d\}\} && (4) \end{aligned}$$

Both Zimmerman's and Ying's algorithms constrain the antecedent of rules to be equal to the change set, and hence only generate rules based on line 4 (i.e., rules of the form $\{a, b, c, d\} \rightarrow X$). At the other end of the spectrum, co-change algorithms only generate rules from the singleton sets in Line 2, such as $\{a\} \rightarrow X$ or $\{b\} \rightarrow X$. In our previous work, we introduced Tarmaq, the most versatile among the existing algorithms [14]. Tarmaq generates rules whose antecedent can be from any of lines 1, 2, 3 or 4. The particular line used is dynamically chosen based on the maximal overlap between the change set C and the transactions that make up the history.

3 Problem Description

Change impact analysis takes as input a set of changed entities (e.g., the files changed in a system), referred to as a *change set* or *query*, and outputs a set of potentially impacted entities. A common strategy for change impact analysis is to use association rule mining to capture the *evolutionary couplings* between such entities. In particular, entities are considered to be coupled if they have changed together in the past. Furthermore, the *strength* of a coupling depends on how frequently the entities have changed together. The stronger the coupling between two entities, the more likely it is that the entities are related to each other, and hence that one is impacted by changes to the other.

The strength of an evolutionary coupling is usually assessed using an *interestingness measure* applied to the association rules. The literature reports over 40 of these measures, which have been applied in a variety of domains [20]. It has been shown that the particular measure chosen has a significant impact over the ranking of rules, and consequently on the quality of the recommendations generated [21–23].

The following four examples illustrate how performance is affected by the two configuration parameters interestingness measure and the filter size, the size of the query and expected outcome, and finally the system characteristics.

Example 1. *Consider the following history $\mathcal{T}$ of transactions:*

$$\mathcal{T} = \big[\{a,c\}, \{b,c\}, \{a,b,c\}, \{a,b,x,y,z\}, \{y,z\}, \{x,y,z\}\big]$$

and the change set $C = \{a, b\}$ where, based on $\mathcal{T}$ and C, the following four rules have been mined.[2] *For each rule three measures,* confidence κ, prevalence ϕ, *and* recall ρ *[21] are given in parentheses. The confidence (recall) measures the ratio between the number of transactions where the antecedent and consequent changed, and the number of transactions where only the antecedent (consequent) changed. Prevalence measures the percentage of transactions in the history where the consequent changed.*

$$\begin{aligned}
a, b \rightarrow c \quad & (\kappa = 1/2,\ \phi = 3/6,\ \rho = 1/3) \\
a, b \rightarrow x \quad & (\kappa = 1/2,\ \phi = 2/6,\ \rho = 1/2) \\
a, b \rightarrow y \quad & (\kappa = 1/2,\ \phi = 3/6,\ \rho = 1/3) \\
a, b \rightarrow z \quad & (\kappa = 1/2,\ \phi = 3/6,\ \rho = 1/3)
\end{aligned}$$

In the example, the confidence values suggest that the four rules are all equally likely to hold, while prevalence suggests that $a, b \rightarrow x$ is inferior to the others, because x is changed only twice in $\mathcal{T}$. On the other hand, recall suggests the opposite, that $a, b \rightarrow x$ is more likely than the others, because the co-occurrence of $\{a, b\}$ and x in $\{a, b, x, y, z\}$ is more significant than the co-occurrence of $\{a, b\}$ and other files that changed more often without both a and b. However, one's intuition matches the suggestion made by ϕ since a and b have a stronger relation with c than with x. This is because c appears also singularly with a and b, while x tends to change together with y and z.

In general, the particular interestingness measure (or combination thereof) used to rank the rules is a parameter of the change recommendation algorithm. There exist a number of such parameters that affect how rules are ranked, and recommendations are generated.

Example 2. *Consider again the history $\mathcal{T}$ of transactions:*

$$\mathcal{T} = \big[\{a, c\}, \{b, c\}, \{a, b, c\}, \{a, b, x, y, z\}, \{y, z\}, \{x, y, z\}\big]$$

and the change set $C = \{a, b\}$. However, assume that transactions larger than three files are discarded from the history when generating rules and calculating their interestingness. Based on C and the filtered history, only the rule $a, b \rightarrow c$ is mined.

Here, x, y, and z will not be recommended, because $\{a, b, x, y, z\}$, the change set containing evidence for their recommendation, has been filtered out of $\mathcal{T}$. Intuitively, filtering out larger transactions from the history avoids generating rules from change sets that contain potentially unrelated files, and is a strategy used by most approaches [11, 12, 14]. However, filtering too aggressively may remove legitimate evidence of evolutionary coupling.

Values for configurable parameters are not the only criteria that can affect the recommendations generated.

Example 3. *Consider the same history $\mathcal{T}$ of transactions:*

$$\mathcal{T} = \big[\{a,c\},\{b,c\},\{a,b,c\},\{a,b,x,y,z\},\{y,z\},\{x,y,z\}\big]$$

and the change sets $C_1 = \{a\}$ and $C_2 = \{a,b\}$. The following rules are mined for C_1 and C_2.

rules for C_1	*rules for C_2*
$a \to b$	$a,b \to c$
$a \to c$	$a,b \to x$
$a \to x$	$a,b \to y$
$a \to y$	$a,b \to z$
$a \to z$	

C_1 and C_2 have different sizes, and hence, when used as queries they provide different amounts of evidence from which to make a recommendation. For example, consider the recommendation of c. In this case, C_1 contains only a in support of the recommendation while C_2 contains both a and b and thus likely provides stronger support. While intuitively, larger queries can be expected to lead to stronger recommendations, queries that are too large may contain potentially unrelated files, degrading the quality of the recommendation.

An inverse argument applies to the *expected outcome*, the set of files that we seek to recommend. First consider the goal of predicting any one file from the expected outcome. Clearly, the larger the expected outcome, the easier the task of predicting one of them correctly. However, our goal is to recommend *all* of the files in the expected outcome. In that case, intuitively, a *smaller* expected outcome should be easier to recommend, as it is easier for a query to provide the necessary supporting evidence.

Finally, characteristics of the change history can affect the precision of a recommendation.

Example 4. *Consider two transaction histories $\mathcal{T}_1$ and $\mathcal{T}_2$ from systems $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Assume that the transactions of $\mathcal{T}_1$ are on average larger than those of $\mathcal{T}_2$ because of differences in development processes. For example, $\mathcal{S}_1$ might be*

Table C.1: Characteristics of the last 30 000 transactions of the evaluated software systems

Software System	Nr. of files	Avg. commit size	History covered (in years)
Cisco Norway	41701	6.20	1.07
Kongsberg Maritime	35111	5.08	15.97
Git	3574	1.95	10.42
HTTPD	10021	4.80	19.78
Linux Kernel	19768	2.14	0.48
LLVM	20745	4.41	2.15
JetBrains IntelliJ	36162	3.38	1.58
Ruby on Rails	5346	2.25	5.78
Subversion	2915	2.76	7.61
Wine	6679	2.47	4.61

Software System	Languages used*
Cisco Norway	C++, C, C#, Python, Java, XML, other build/config
Kongsberg Maritime	C++, C, XML, other build/config
Git	C (45%), shell script (35%), Perl (9%), 14 other (11%)
HTTPD	XML (56%), C (32%), Forth (8%), 19 other (4%)
Linux Kernel	C (94%), 16 other (6%)
LLVM	C++ (71%), Assembly (15%), C (10%), 16 other (4%)
JetBrains IntelliJ	Java (71%), Python (17%), XML (5%), 26 other (7%)
Ruby on Rails	Ruby (98%), 6 other (2%)
Subversion	C (61%), Python (19%), C++ (7%), 15 other (13%)
Wine	C (97%), 16 other (3%)

* language information from `http://www.openhub.net`, percentages for the industrial systems are not disclosed.

developed with an agile process, where small change sets are committed frequently while $\mathcal{S}_2$ was developed with a less nimble process, where large change sets are committed less often.

In this last example, queries and expected outcomes for $\mathcal{T}_1$ are likely to be smaller than those for $\mathcal{T}_2$. Thus entailing the implications discussed in Example 3, and also potentially differing impacts from transaction filtering which was described in Example 2.

4 Empirical Study

We perform a large empirical study to assess the extent to which the quality of change recommendations generated using targeted association rule mining is affected by (1) the values of various parameters of the mining algorithm, (2) characteristics of the change set used to derive the recommendation, and (3) characteristics of the system's change history. We use as a reference mining approach from our previous work, which has proven to perform consistently better than the previous state-of the art for software change impact analysis [14]. Our study investigates the performance of targeted association rule mining in the context of several software-systems, and several parameters configurations. Specifically, we investigate the following four research questions:

RQ 1. *To what extent does the interestingness measure affect the precision of change recommendation?*

RQ 2. *To what extent does the size limit used to filter the transactions of the history affect the precision of change recommendation?*

RQ 3. *To what extent do query size and expected outcome size affect the precision of change recommendation?*

RQ 4. *To what extent does the average commit (transaction) size of the history affect the precision of change recommendation?*

The remainder of this section details our evaluation setup, and is organized as follows: in subsection 4.1 we describe the software-systems included in the study. Sections 4.2 and 4.3 describe the interestingness measures and the history filtering. Subsection 4.4 describes two central concepts for the evaluation, *query generation* and *query execution*. Subsection 4.5 explains the generation of change recommendations. Finally, subsection 4.6 explains how we measure performance.

4.1 Subject Systems

To assess change recommendation in a variety of conditions, we selected ten large systems having varying size and transaction frequency. Two of these systems come from our industry partners, Cisco Norway and Kongsberg Maritime (KM). Cisco Norway is the Norwegian division of Cisco Systems, a worldwide leader in the production of networking equipment. In particular, we consider a software product line for professional video conferencing systems developed by Cisco Norway. KM is a leading company in the production of systems for positioning, surveying, navigation, and automation of merchant vessels and offshore installations. Specifically, we consider a common software platform KM uses across various systems in the maritime and energy domain.

The other eight systems are well known open-source projects. Table C.1 summarizes descriptive characteristics of the software systems used in the evaluation. The table shows that the systems vary from medium to large in size, with up to forty thousand different files committed in the transaction history. For each system, we considered the 30 000 most recent transactions (*commits*). This value represents a balance between too short a history, which would lack sufficient connections, and too long a history, which is hard to process efficiently and can contain outdated couplings caused by, for example, architectural changes. Across all ten systems, 30 000 transactions covers a significantly different development time span, ranging from almost 20 years in the case of HTTPD, to 6 months in the case of the Linux kernel. Most of the systems are heterogeneous, being developed in more than one programming language. Finally, we note that the median commit size for all the selected systems is one.

4.2 Interestingness Measures

Mining of change recommendations often uses or combines the *support* and *confidence* measures from the data mining community to separate interesting from uninteresting rules [11, 12, 14, 24]. However, as introduced in section 3, over 40 interestingness measures have been defined in the literature to measure the strength of the evolutionary couplings mined using association rules. These measures are usually defined based on a probabilistic interpretation of the occurrence in the history of the rules antecedent and consequent. For example, given the rule $A \rightarrow B$ the probability $P(A)$ is the percentage of transactions from the history that include A, while the probability $P(A, B)$ is the percentage of transactions in the history that contain both A and B. Therefore, the interestingness of the rule $A \rightarrow B$ is usually defined as a function of $P(A)$, $P(B)$, and various combinations thereof obtained through negations, fractions, and conditional operators. In this paper, we consider 39 interestingness measures commonly used in several data mining and machine learning applications. Due to space limitations, we only report their names in Table C.2, and refer the reader to the original sources for the definitions [25, 26].

4.3 History Filtering

Several approaches for mining change recommendations start by filtering the history to remove transactions larger than a given size. This common heuristic seeks to avoid mining from transactions that do not contain relevant in-

Table C.2: Overview of the 39 interestingness measures considered in our study

#	Interestingness Measure		
1	Added Value	21	Least Contradiction
2	Casual Confidence	22	Leverage
3	Casual Support	23	Lift
4	Collective Strength	24	Linear Correlation Coefficient
5	Confidence	25	Loevinger
6	Conviction	26	Odd Multiplier
7	Cosine	27	Odds Ratio
8	Coverage	28	One Way Support
9	Descriptive Confirmed Confidence	29	Prevalence
10	Difference Of Confidence	30	Recall
11	Example and Counterexample Rate	31	Relative Risk
12	Gini Index	32	Sebag Schoenauer
13	Imbalance Ratio	33	Specificity
14	Interestingness Weighting Dependency	34	Support
15	J Measure	35	Two Way Support
16	Jaccard	36	Varying Rates Liaison
17	Kappa	37	Yules Q
18	Klosgen	38	Yules Y
19	Kulczynski	39	Zhang
20	Laplace Corrected Confidence		

formation on the evolutionary coupling of files, such as in the case of license updates or refactoring [11, 12, 24, 27]. Removing transactions from the history also has the effect of significantly speeding up the rule generation process.

This paper considers seven different transaction filtering sizes: 2, 4, 6, 8, 10, 20, and 30. Note that 30 is the threshold used in the work of Zimmermann et al. [11]. Starting from this value, we progressively consider more restrictive filtering. For each filter size s, we generate a filtered history H_s, from which we mine the association rules used to generate recommendations. In addition, we also consider the unfiltered history H. Since H can be thought of as filtered with an infinite filter size, we simply refer to these as *eight* transaction filtering sizes in the rest of the paper.

4.4 Query Generation and Execution

A challenge in evaluating change recommendation techniques is what "*gold standard*" can be used to compare the generated recommendation against. A common strategy [11–13] is to take a transaction T from the change history and randomly partition it into a non-empty query Q and a non-empty expected outcome $E \overset{\text{def}}{=} T \setminus Q$. Since the files in a transaction are considered to be logically coupled [15], this stategy provides a suitable gold standard. The evaluation then assesses how well a change recommendation technique can recover expected outcome E from query Q using the transactions that came before T in the change history.

From the history of each system, we randomly sample 1500 transactions,[3] which are used to generate 1500 queries (and their related expected outcomes). Each query is executed for each of the 39 interestingness measures reported in Table C.2, using each of the eight filtering sizes introduced in subsection 4.3. This setup yields a total of $1500 \cdot 39 \cdot 8 = 468\,000$ data points for each system, where each data point is a recommendation for a query.

4.5 Generating Change Recommendations

All queries are executed using Tarmaq, the targeted association rule mining algorithm we introduced in previous work [14]. Recall from section 2 that executing a query Q creates a set of association rules. In order to efficiently generate recommendations, Tarmaq only considers rules whose consequent contains a single file [14]. Generating a change recommendation for Q consists of sorting the rules generated for Q according to their interestingness score, and returning the ranked list of the consequents of the rules. We only consider the largest interestingness score for each consequent, i.e., we do not use rule aggregation strategies, such as the ones we proposed in previous work [26].

4.6 Performance Measure

To evaluate a recommendation we use *average precision* (AP), which is commonly used in Information Retrieval to assess the quality of a ranked list [28]:

Definition 1 (Average Precision). *Given a recommendation* R*, and an expected outcome* E*, the average precision of* R *is given by:*

$$AP(R) \overset{\text{def}}{=} \sum_{k=1}^{|R|} P(k) * \triangle r(k)$$

[3] For a normally distributed population of 30 000, a minimum of 651 samples is required to achieve a 99% confidence level with a 5% confidence interval. Since we do not know the distribution of transactions, we correct the sample size to the number needed for a non-parametric test to have the same ability to reject the null hypothesis. The correcting is done using the Asymptotic Relative Efficiency (ARE). As AREs differ for various non-parametric tests, we choose the lowest coefficient, 0.637, yielding a minimum sample size of $651/0.637 = 1022$ transactions. Hence, sampling 1500 transactions is more than sufficient to achieve a 99% confidence level with a 5% confidence interval.

Table C.3: Example of average precision calculation

Consider $\{c, d, f\}$ as expected outcome in the following list:

Rank (k)	File	$P(k)$	$\triangle r(k)$
1	c	1/1	1/3
2	a	1/2	0
3	f	2/3	1/3
4	g	2/4	0
5	d	3/5	1/3

$$AP = 1/1 \cdot 1/3 + 1/2 \cdot 0 + 2/3 \cdot 1/3 + 2/4 \cdot 0 + 3/5 \cdot 1/3 \approx 0.75$$

Table C.4: *Commit size* frequency and cumulative percentage of all transactions considered.

commit size	1	2	3	4	5	6	7	8	9	10	(10,20]	(20,30]	>30
frequency	157636	48903	23801	13767	8561	5546	3829	2814	2103	1611	6536	1778	2717
cumulative %	56.4%	73.9%	82.4%	87.3%	90.4%	92.4%	93.7%	94.7%	95.5%	96.1%	98.4%	99.0%	100.0%

where $P(k)$ is the precision calculated on the first k files in the list (i.e., the fraction of correct files in the top k files), and $\triangle r(k)$ is the change in recall *calculated only on the $k-1^{th}$ and k^{th} files (i.e., how many more correct files were predicted compared to the previous rank).*

Since we consider only rules with a single consequent, $\triangle r(k)$ will always be equal to either zero or $1/|E|$, because a rank either does not contain a file from expected outcome E, or it contains exactly one file from E. Table C.3 illustrates the computation of AP, $P(k)$, and $\triangle r(k)$ given the ranked list $[c, a, f, g, d]$ and the expected outcome $\{c, d, f\}$.

As an overall performance measure for a group of factors (e.g., a given filtering size and interestingness measure) we use the *mean average precision* (MAP) computed over all the queries executed using the given factor combination.

5 Results

This section presents the results of the study described in section 4. A replication package is provided online.[4]

We begin by analyzing a key descriptive statistic, the *commit size* distribution, which is shown in Table C.4. Because the majority (90.4%) of the commits contains less than six files, we focus the remainder of our analysis on this dominant subset of the data.

[4] `https://evolveit.bitbucket.io/publications/ase2016/replication/`

Table C.5: ANOVA Model - Explanatory variables and their pair-wise interactions, ordered by F-value, which provides a measure of the significance of the variable/interaction.

Explanatory Variable	Df	Mean Sq	F-value	*p*-value
expected outcome size	1	2244	15930	<0.0001
program	9	1643	11667	<0.0001
filter size	1	1411	10018	<0.0001
measure	38	452	3212	<0.0001
query size	1	130	923	<0.0001
expected outcome sz:filter size	1	122	864	<0.0001
program:query size	9	55	390	<0.0001
program:expected outcome size	9	47	332	<0.0001
program:filter size	9	27	194	<0.0001
query size:filter size	1	21	149	<0.0001
query size:measure	38	21	145	<0.0001
query size:expected outcome sz	1	20	142	<0.0001
expected outcome size:measure	38	5	38	<0.0001
filter size:measure	38	5	33	<0.0001
program:measure	342	4	28	<0.0001

5.1 Analysis of Explanatory Variables

Central to our analysis is an ANOVA, used to explore the significance of the five explanatory variables *program*, *filter size*, *expected outcome size*, *interestingness measure* (or *measure* for short), and *query size*, together with all ten of their pairwise interactions. The Q-Q Plot of the residuals (left out) finds that these residuals follow a sufficiently normal distribution, especially considering ANOVA's performance in the presence of large data sets such as the nearly five million data points considered here.

The model is shown in Table C.5, with terms ordered by their F-value. Even though all fifteen terms are highly statistically significant, the five variables make a larger contribution than the interaction terms (i.e., have larger F-values).

5.2 Impact of Interestingness Measures

Using the ANOVA, we continue our analysis by considering the first research question:

RQ 1 *To what extent does the interestingness measure affect the precision of change recommendation?*

To visually explore the influence of *interestingness measures*, Figures C.1–C.4 show interaction plots of *measure* with respectively *program*, *query size*, *expected outcome size*, and *filter size* (where *inf* indicates no filtering). The overall pattern, evident in these graphs, is that *measure* is largely independent of *program*, *query size*, *expected outcome size*, and *filter size*. Statistically, the interactions are significant primarily due to a few interestingness measures that "buck the trend" (producing line crosses in the plots). This visually evident "mostly independent" observation is supported by the comparatively low F-values for the four interactions involving "*measure*" (they are four of the five smallest in the model shown in Table C.5).

The low F-values and the interactions plots support the notion that there are consistent best *measures* for change recommendation based on evolutionary coupling. Not surprisingly, there is no single best *measure*. Tukey's honestly significant difference (HSD) test finds that eleven measures populate the top equivalence class. These are shown in Table C.6. Note that even though their MAP values are slightly different, the measures in this class are not statistically different from each other. Also observe that Table C.6 includes the classic measures *confidence* and *support*. Their presence reinforces a result from the recent work of Le and Lo [25]. While their work considers a different problem (the effect of different interestingness measures in rule-based specification mining), they too conclude that the standard measures work well. Thus in summary for RQ1, we find that to a large extent *measure*'s influence on average precision is consistent across differences in other variables and

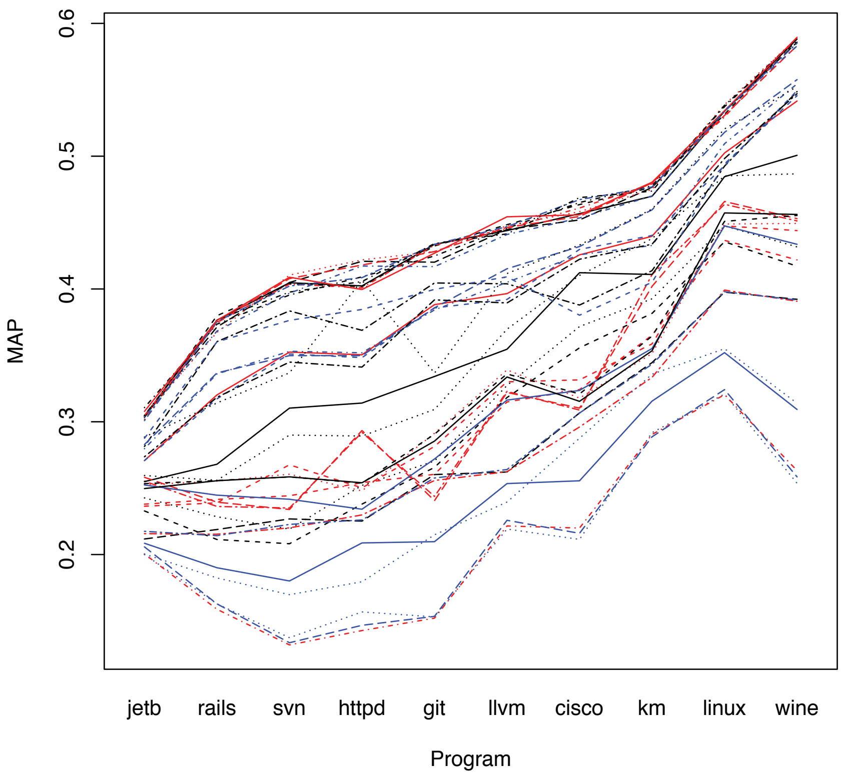


Fig. C.1: Interaction plot of *measure* and *program*. The *measures* are shown unlabeled to avoid cluttering the plot.

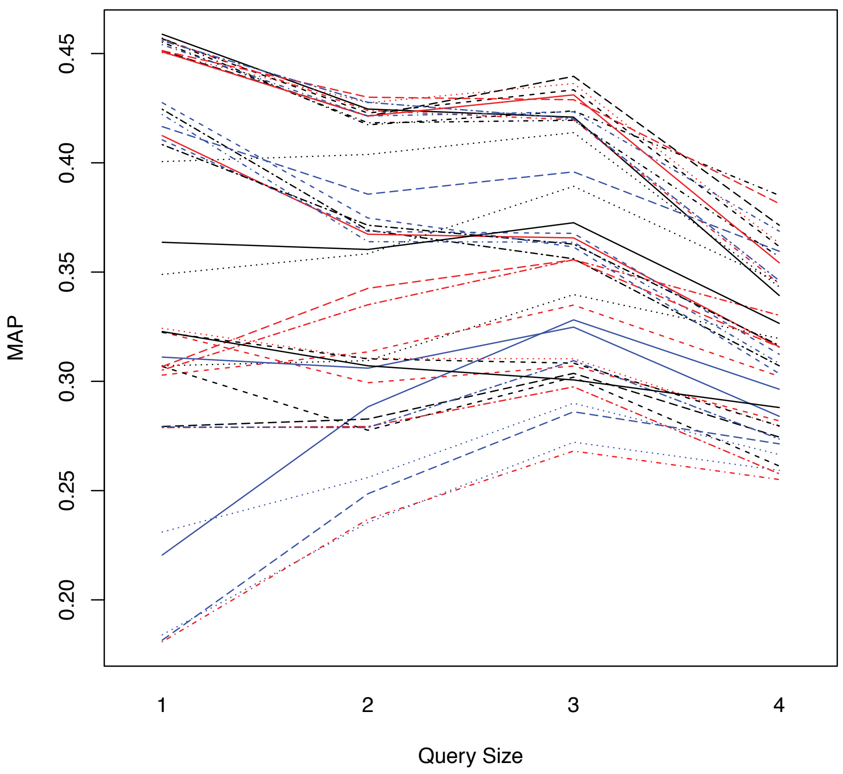


Fig. C.2: Interaction plot of *measure* and *query size*. The *measures* are shown unlabeled to avoid cluttering the plot.

that the traditional measures are top performers.

5.3 Impact of Transaction Filtering

Next, we turn to answering our second research question:

RQ 2 *To what extent does the size limit used to filter the transactions of the history affect the precision of change recommendation?*

The results for this question are rather surprising. When filtering the history it is common to remove large commits as they are assumed to reflect licencing changes and alike. Prior work has typically used 30 as a cut off, while some experiments have used values as high as 100 [11, 12].

One might wonder if *filter sizes* 10, 20, or 30 yield any differences, because the average commit sizes for the ten systems studied are smaller than these filter sizes. The answer can be in Figure C.4, which shows that no filtering (filter size *inf*) performs worse than filtering commits larger than 30, which in turn performs worse than filtering commits larger than 20, and similarly for filter size 10. Analysis using Tukey's HSD also shows that the MAPs continue to significantly decrease for filter sizes 10, 20, 30, and no filtering.

For the eight *filter size*s studied, Figure C.5 shows the interaction between *filter size* and *program* for two subsets of the data. On the left, the data for all *measures* is shown, while on the right, the data for the top group of measures identified in Table C.6 is shown. It is clear from this figure that smaller commits contain much more useful information than larger commits as the best value for most systems occurs at a *filter size* of only four or six when considering all measures, and at a *filter size* of six to ten for the top group of measures.

Considering only the top group of measures, Tukey's HSD test, shown in Table C.7, groups *filter size* six and eight in the top equivalence class. Thus, in answer to RQ2, aggressive history filtering appears to retain only high value commits that support the creation of high quality association rules.

This is a noteworthy finding, as it suggests that filtering should be applied much more aggressively. Although small commits capture stronger couplings, prior work has not exploited this fact. For example, as discussed in section 2, Rose [11] only generates association rules whose antecedent is equal to the query. In particular, this means that the algorithm can not generate rules from transactions smaller than the query. In contrast, the more recent algorithm Tarmaq [14] can exploit partial matches between the query and historical transactions, and can thus be used when considering only small high-focus commits.

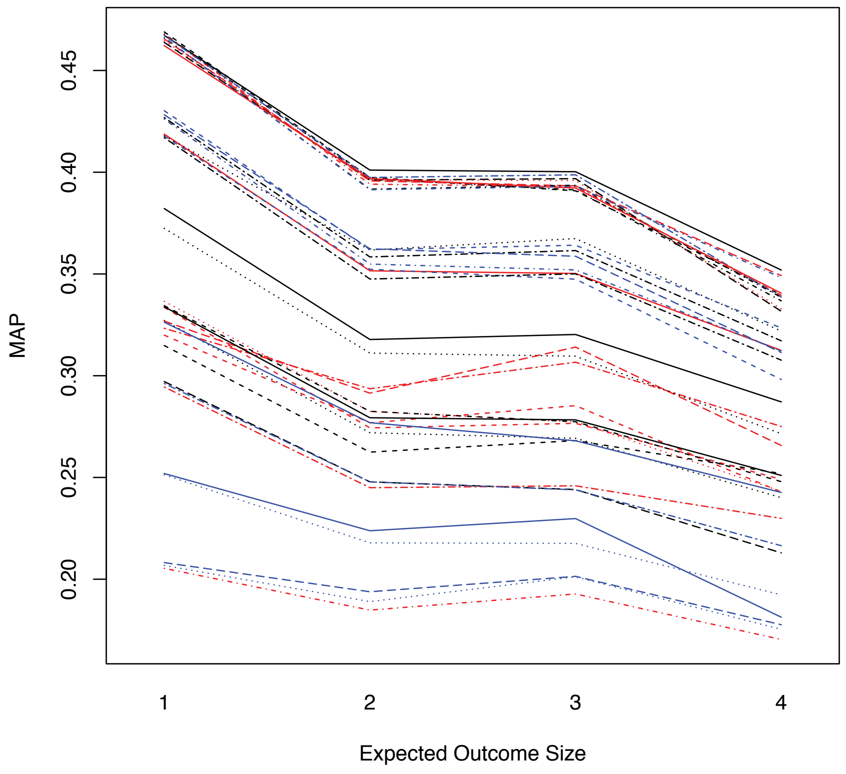


Fig. C.3: Interaction plot of *measure* and *expected outcome size*. The *measures* are shown unlabeled to avoid clutter.

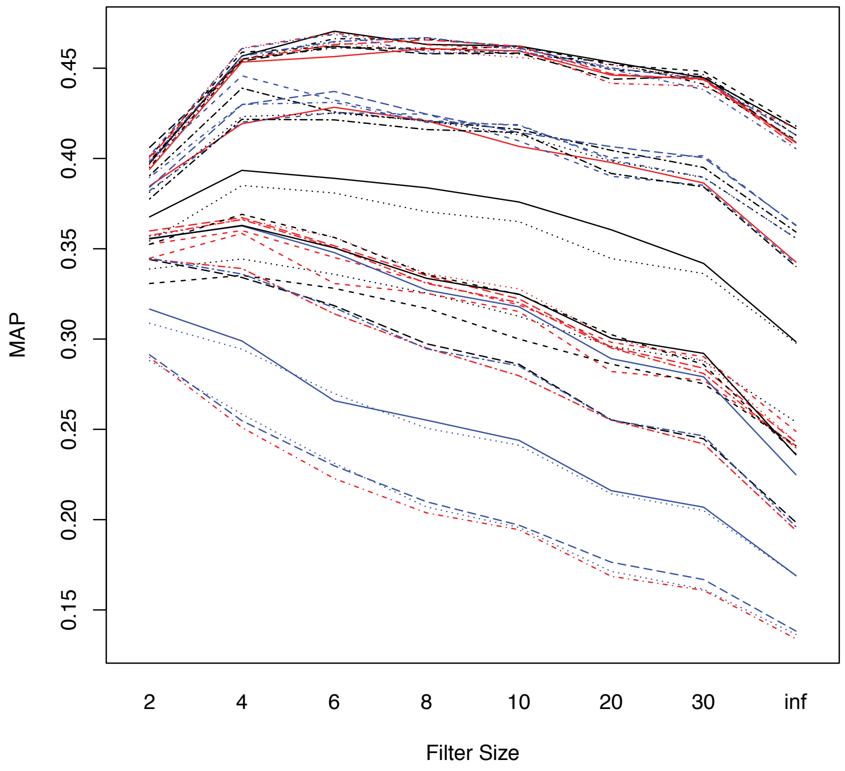


Fig. C.4: Interaction plot of *measure* and *filter size*. The *measures* are unlabeled to avoid cluttering the plot.

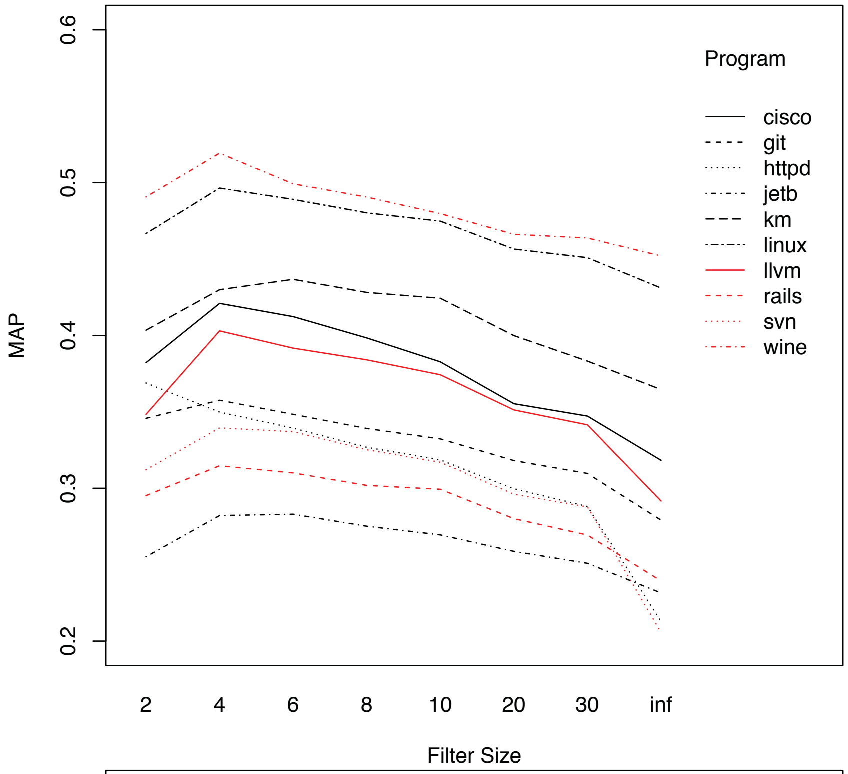
5.4 Impact of Query Size and Expected Outcome Size

Our third research question considers the impact of the query and the expected outcome:

RQ 3 *To what extent do query size and expected outcome size affect the precision of change recommendation?*

Because all the explanatory variables *and* all their interactions are statistically significant, the resulting coefficient equation is very complex (it includes almost 100 terms!). Rather than state this equation, we zoom in on a few of its key terms. Specifically those involving *query size*, *expected outcome size*, and *commit size*, which is the sum of *query size* and *expected outcome size*. These three include three significant interactions and thus the interpretation of even just this subset is complex.

Fortunately the practical range of these explanatory variables is limited (as shown in Table C.4, where 90% of the data is covered by commit sizes less than or equal to 5). The use of a small range of values makes it is viable to enumerate all possible combinations. Table C.8 unravels the interactions for the various combinations of *query size* and *expected outcome size* and uses color to indicate entries with the same *commit size*. For example, the hardest case (where the MAP value gets the lowest contribution) is for a *query size* of 1 and



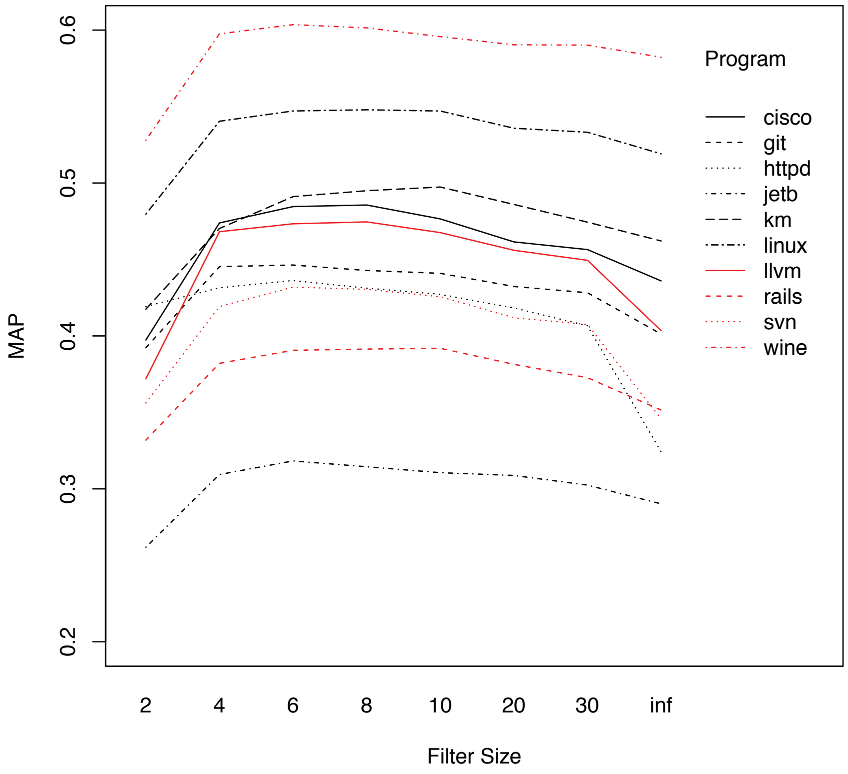


Fig. C.5: Interactions of *filter size* and *program*. To the left for all *measures*, and to the right for the top group of *measures*.

Table C.6: Top group of Tukey's HSD for *measure*

interestingness measure	MAP	group
casual confidence	0.446	a
klosgen	0.446	a
descriptive confirmed confidence	0.446	a
added value	0.445	a
collective strength	0.445	a
loevinger	0.445	a
confidence	0.444	a
leverage	0.443	a
example and counterexample rate	0.443	a
difference of confidence	0.443	a
support	0.442	a

Table C.7: Tukey's HSD for *filter size* (for the top *measures*).

	filter size	
size	mean	group
6	0.464	a
8	0.463	ab
10	0.459	bc
4	0.457	c

an *expected outcome size* of 4. This high-challenge level is expected as this case aims to predict the largest of the outcomes based on minimal information.

The table makes it possible to look at trends. Starting with *commit size*, following the diagonal with green cells, which have a *commit size* of five, we see that the prediction gets easier (the contribution to MAP increases) as the *expected outcome size* decreases, and the *query size* increases. This same pattern is seen with the other commit sizes (other colors). A similar pattern is also clear for a fixed *query size* (any given column of the table). In this case, the prediction gets more difficult (the contribution to MAP decreases) as *expected outcome size* grows.

Finally, for a fixed *expected outcome size* (a given row in the table) the trend is less clear: one might expect the prediction to get easier as *query size* increases, because there is more information to build on. We can indeed see this pattern in the row for *expected outcome size* of three. However, the row where *expected outcome size* is two is approximately flat, and most surprisingly, the top row where *expected outcome size* is one shows a clearly decreasing trend. Thus, the data shows that the prediction of a single result gets more difficult (the contribution to MAP decreases) as *query size* grows.

We can not completely explain this decrease, but conjecture that it is an effect related to small commits (such as those shown in yellow and blue) being more focussed than larger commits. In other words, larger commits are more likely to introduce noise into the recommendation. It may also be an artifact of using coarse-grained (file level) change data, and thus the expected trend would be visible with more fine-grained (method level) change data. We plan to investigate this phenomena in more detail in our future work.

5.5 Impact of Average Transaction Size

Finally, we consider our last research question:

RQ 4 *To what extent does the average commit (transaction) size of the history affect the precision of change recommendation?*

To analyze this question we perform a linear regression using the *average commit size* to predict the MAP value for each system. This data is shown in Table C.9. The regression finds no significant correlation between the *average commit size* and MAP. Thus in answer to RQ4: the *average commit size* in the change history has no impact on the precision of change recommendation. In retrospect, considering how Table C.4 shows that the frequency of *commit sizes* is rather skewed, the finding that the *average commit size* is not a good predictor of the algorithm's precision should come as no surprise.

5.6 Threats to Validity

Commits as a basis for evolutionary coupling: The evaluation presented in this paper is grounded in predictions made from analyzing patterns in change histories. However, as pointed out by Herzig et al. [29, 30], the transactions in the change histories could contain unrelated files, or could miss related files added in subsequent transactions. In our case, the software systems studied except KM use Git for version control, which provides developers with tools for amending commits and rewriting history. Therefore, we argue that the impact of incomplete or entangled transactions is less signif-

Table C.8: Relative impact of *query size* and *expected outcome size* and their interactions. The colors indicate the commit size, as shown in the legend on the right.

expected outcome size	*query size* 1	2	3	4		*commit size*
1	🟨 10	🟦 9	🟪 8	🟩 5		🟨 2
2	🟦 5	🟪 5	🟩 4			🟦 3
3	🟪 2	🟩 3				🟪 4
4	🟩 1					🟩 5

icant in our case than it would be using other version control systems such as SVN or CVS. In our related work, we also discuss methods for grouping related commits that are orthogonal to our approach, and could be thus used to refine the change history.
Variation in software systems: We conducted our experiments on two industrial systems and eight large open source systems. These systems vary considerably in size and frequency of transactions (commits), which should provide an accurate picture of the performance in various contexts (see Table C.1). However, despite our careful choice, we are likely not to have captured all possible variations.
Random sampling errors: Our experiment is based on taking a large number of random samples from the change history of each system. Although we use uniform random sampling, there is the possibility that our samples do not accurately represent the actual change history, for example the distribution of transaction sizes considered in the sample may be different full history. We address this particular issue applying a chi-squared test to validate that our samples are representative of the population (the change histories). An alternative approach would be to use a stratified sampling strategy where one extracts samples of each transaction size in proportion to their frequency in the complete history. The results of the chi-squared test indicate that such an alternative strategy is not necessary.
Implementation: We have implemented the experiments using Ruby and thoroughly tested all algorithms and measures studied in this paper. We also performed the statistical analysis using standard methods provided by R. However, we can not guarantee the absence of implementation errors, which may affect our results.

Table C.9: Average *commit size* and MAP for each *program*.

program	average *commit size*	MAP
cisco	6.20	0.375
git	1.96	0.328
httpd	4.80	0.311
jetbrains	3.39	0.263
km	5.08	0.408
linux	2.15	0.468
llvm	4.42	0.360
rails	2.25	0.288
subversion	2.77	0.302
wine	2.47	0.482

6 Related Work

Recent research highlighted that the performance of data mining algorithms is impacted by their configuration parameters [31]. A common strategy for tuning these parameters consists in optimizing their values over half of the test set (*inner* validation), and then using the other half to assess the performance of the optimal parameters (*outer* validation) [32]. While this strategy is useful to fine-tune the parameters for a particular type of data, it does not provide insights on how the algorithm's performance is affected by the parameters [33]. In the context of association rule mining, several authors have noted the need to investigate the impact of parameter settings on the quality of the generated rules [34–36].

Therefore, in this paper, we investigate the extent to which the quality of change recommendation via association rule mining is affected by three factors (1) the values selected for various parameters of the mining algorithm, (2) characteristics of the change set used to derive the recommendation, and (3) characteristics of the system's change history for which recommendations are generated. In the rest of this section, we distinguish related work on these three aspects, focusing on the area of software maintenance and evolution.

Parameters in Association Rule Mining: In general, association rule mining algorithms differ from each other in the data structures used to represent transactions, and the strategy used to select transactions relevant to a given query [37]. However, the majority of such algorithms are characterized by similar parameters. Among those, this paper focuses on the maximum size of transactions used to generate rules (*transaction filtering size*), and on the metric measuring the strength of evolutionary couplings inferred by those rules (*interestingness measure*). In the context of software change impact analysis, several studies remark on the importance of discarding from the history large change sets that are likely to contain unrelated files. For example, Kagdi et al [24], Zimmermann et al. [11] and Ying et al. [12] propose filtering transactions larger than 10, 30, and 100 items, respectively. However, these authors generally do not report how such threshold values have been chosen, nor do they explore the impact of alternative values on recommendation precision. Several authors have also remarked that selecting the right interestingness measure for a problem domain can significantly affect recommendation accuracy [20, 22, 23, 25]. In particular, Le and Lo compared 38 measures in the context of rule-based specification mining, highlighting the need to look beyond standard support and confidence to find interesting rules [25]. We are not aware of similar studies carried out in the context of software change impact analysis.

Characteristics of the Change Set: Targeted association rule mining uses the query to drive the generation of rules [16]. In general, particular characteristics of the query can effect the quality of recommendations. For example, in the context of software change impact analysis, we identified in previous work a particular class of queries for which the most common targeted association rule mining approaches cannot generate recommendations [14]. Note that it is common practice to validate targeted association rule mining approaches by sampling random queries from the change history [11, 12]. While this strategy ensures that the approach is evaluated on a variety of change sets from the history, authors in general do not consider how query size might affect the quality of the recommendations. Among others, Hassan and Holt investigated the effectiveness of evolutionary coupling in predicting change propagation effects resulting from source code changes, but did not evaluate whether the size of transactions in the history affected the quality of the predictions generated [19].

Characteristics of the Change History: Several past studies have proposed strategies to group transactions in the change history [11, 13, 38]. The reason for doing so is that a developer might (accidentally) commit an incomplete transaction, and update the remaining files related to the same change in a separate transaction. As a consequence, entities that are logically coupled via the same change might become spread across several transactions in the change history. Nevertheless, in modern version control systems transactions are stashed in the user's local repository and can be amended before they are finalized at a later stage, thereby reducing the risk of committing incomplete transactions. In contrast, the question whether properties of the change history, such as average commit size and frequency, affect the quality of change recommendation is less studied. In this direction, German carried out an empirical study on several open source projects, finding that the revision histories of most systems contains mostly small commits [39]. Alali et al. also investigated the total number of lines modified in the files and the total number of hunks (continuous groups of lines) that were changed [40]. Kolassa et al. performed a similar study on commit frequency, reporting an average inter-commit time of about three days [41]. However, none of these studies investigate how characteristics of the change history affect the quality of the change recommendation.

7 Concluding Remarks

Conclusions: Association rule mining is an automated, unsupervised, learning technique that has been successfully used to analyze a system's change history and uncover *evolutionary coupling* between its artifacts. These evolutionary couplings can be used to *recommend* artifacts that are potentially *impacted by a given set of changes* to the system. Doing so can help developers address the increasing entropy caused by repeated software maintenance and evolution.

In this paper, we present a series of experiments using the change histories of two large industrial systems and eight large open source systems. For each system, we randomly extract a representative sample of the transactions from the change history, randomly split each of these transactions in two parts, a query and an expected outcome, and analyzed how several parameters affect the prediction of the expected outcome based on the query.

We draw the following conclusions: first, our analysis shows that the impact of *interestingness measure* is largely independent of the particular *program*, *query size*, and *expected outcome size*. This means that it is possible to identify the best measures for change recommendations based on evolutionary coupling. To do so, we clustered measures using Tukey's HSD where the measures in each cluster are not statistically different from each other. We find that the standard measures *confidence* and *support* are in the top equivalence class; thus statistically no other measure provides better performance than these two. This result is in line with the earlier findings by Lo and Le [25], who conducted a related study of interestingness measures for rule-based specification mining.

Second, learning from a change history that is filtered to remove transactions larger than six to eight files yields the best results, regardless of *program*, *query size*, and *expected outcome size*. Furthermore, the average precision produced by higher thresholds is significant worse.

Third, for smaller commits (up to a transaction size of five, which includes just over 90% of all transactions), the smaller the *expected outcome*, the higher the average precision. This suggests that for relatively small commits the less information you want to predict, the better the results. When we extend the analysis to larger commits (which are the minority), larger expected outcomes lead to higher precision. Combined these two observations suggest that predicting a few missed files is easy, while predicting a large number of missed files from limited information is hard. A similar pattern is seen when comparing smaller and larger commit sizes.

Fourth, for the smaller commits one would expect that larger queries would produce better average precision. Our data only partially supports this expectation. Looking at the rows of Table C.8 two effects are competing with each other. A simplified version of the coefficient equation used to fill in this table includes both $+$*query size* and $-$*query size*^{2}. The data in the table reflects these two factors. Initially the linear term dominates and increased *query size* increases the average precision. However, for larger values of *query size* the negative quadratic term dominates and the average precision is reduced. While there is insufficient data to establish a trend, the peak value appears to be a function of the *expected outcome size*. In the first row the value is less than or equal to one, for the second row it is two while for the third and fourth rows it is greater than two. Clearly, this effect warrants future research consideration.

Guidelines: From our conclusions, we derive the following practical guidelines for applying association rule mining in the context of software change recommendation: (1) stick with default interestingness measures, as they are in the top perform class and have a long proven track record, (2) learn from a subset of the change history that includes only the smaller transaction sizes to avoid noise. We have good experience with transaction sizes up to about eight, but these values may depend on the actual transaction sizes in the particular history being analyzed.

Future Work: We consider the following directions for future extension of this work: (1) *Use of fine-grained co-change information.* Although our algorithms are granularity agnostic, we expect interesting differences in the *change patterns* used to mine evolutionary coupling at different levels of granularity. For instance, consider how multiple method-level changes in the same file get abstracted into one change at the file-level. These differences will likely have an effect on parameter values, such as the transaction filter size. (2) *Language-specific change recommendations.* The change recommendation mining in this paper is independent of the programming language of the artifacts in the change history. This has advantages in the context of heterogeneous systems, but it also potentially increases noise. Consider a software development project that consists of a front-end written in Java and a back-end written in C. In this case, front-end Java developers are not likely to benefit from recommendations involving the C code. In contrast, *full-stack integrators*, responsible for connecting the front and back end, would benefit from multi-language recommendations. In this context, there is value in being able to provide *language-specific change recommendations*, i.e., recommendations that are based on filtering the change history for artifacts of a particular type. (3) *Impact of software development styles.* The third area of related work will involve a deeper analysis of the impact of various types of software systems and their software development styles on the quality of change recommendations. Aspects that may play a role here include frequency of commits, tendency to piggyback/tangle commits, and code ownership. The question is how to classify systems based on these factors. An initial study might compare industrial software systems with open source systems, as there are often clear differences between the two styles. With that comparison in mind, it is of interest to observe that the two industrial systems that were considered in this study could not be distinguished from the open source systems based on the interaction plots (figures C.1–C.5), nor could be distinguished in terms of MAP or using Tukey's HSD analysis. However, this observation is based on a very small sample size. A more complete comparison should consider a larger sample of industrial systems, unfortunately acquiring such may not be an easy task. We welcome suggestions for tackling this challenge.

Acknowledgement: This work is supported by the Research Council of Norway through the EvolveIT project (#221751/F20) and the Certus SFI (#203461/030). Dr. Binkley is supported by NSF grant IIA-1360707 and a J. William Fulbright award.

References

[1] G. Canfora and L. Cerulo, “Impact Analysis by Mining Software and Change Request Repositories,” in *International Software Metrics Symposium (METRICS)*. IEEE, 2005, pp. 29–37. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1509307http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1509307

[2] M.-A. Jashki, R. Zafarani, and E. Bagheri, “Towards a more efficient static software change impact analysis method,” in *ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE)*. ACM, 2008, pp. 84–90. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1512475.1512493

[3] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, “Chianti: a tool for change impact analysis of java programs,” in *ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2004, pp. 432–448. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1035292.1029012

[4] M. B. Zanjani, G. Swartzendruber, and H. Kagdi, “Impact analysis of change requests on source code based on interaction and commit histories,” in *International Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 162–171. [Online]. Available: http://dl.acm.org/citation.cfm?id=2597096http://dx.doi.org/10.1145/2597073.2597096

[5] S. Bohner and R. Arnold, *Software Change Impact Analysis*. CA, USA: IEEE, 1996.

[6] A. R. Yazdanshenas and L. Moonen, “Crossing the boundaries while analyzing heterogeneous component-based software systems,” in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011, pp. 193–202. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2011.6080786http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6080786

[7] A. Podgurski and L. Clarke, “A formal model of program dependences and its implications for software testing, debugging, and maintenance,” *IEEE Transactions on Software Engineering*, vol. 16, no. 9, pp. 965–979,

1990. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=58784

[8] S. Eick, T. L. Graves, A. Karr, J. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 1–12, 2001. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=895984

[9] R. Robbes, D. Pollet, and M. Lanza, "Logical Coupling Based on Fine-Grained Change Information," in *Working Conference on Reverse Engineering (WCRE)*. IEEE, 2008, pp. 42–46. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4656392

[10] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD International Conference on Management of Data*. ACM, 1993, pp. 207–216. [Online]. Available: http://portal.acm.org/citation.cfm?doid=170035.170072

[11] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1463228

[12] A. T. T. Ying, G. Murphy, R. T. Ng, and M. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1324645

[13] H. Kagdi, S. Yusuf, and J. I. Maletic, "Mining sequences of changed-files from version histories," in *International Workshop on Mining Software Repositories (MSR)*. ACM, 2006, pp. 47–53. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1137983.1137996

[14] T. Rolfsnes, S. Di Alesio, R. Behjati, L. Moonen, and D. W. Binkley, "Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis," in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, mar 2016, pp. 201–212. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7476643

[15] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 1998, pp. 190–198. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=738508

[16] R. Srikant, Q. Vu, and R. Agrawal, "Mining Association Rules with Item Constraints," in *International Conference on Knowledge Discovery and Data Mining (KDD)*. AASI, 1997, pp. 67–73.

[17] T. Ball, J. Kim, and H. P. Siy, "If your version control system could talk," in *ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering*, 1997. [Online]. Available: http://csalpha.ist.unomaha.edu/{~}hsiy/research/visual.pdf

[18] D. Beyer and A. Noack, "Clustering Software Artifacts Based on Frequent Common Changes," in *International Workshop on Program Comprehension (IWPC)*. IEEE, 2005, pp. 259–268. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1421041

[19] A. E. Hassan and R. Holt, "Predicting change propagation in software systems," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2004, pp. 284–293. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1357812

[20] L. Geng and H. J. Hamilton, "Interestingness measures for data mining," *ACM Computing Surveys*, vol. 38, no. 3, sep 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1132960.1132963

[21] M. Kamber and R. Shinghal, "Evaluating the Interestingness of Characteristic Rules," in *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1996, pp. 263–266.

[22] P.-N. Tan, V. Kumar, and J. Srivastava, "Selecting the right objective measure for association analysis," *Information Systems*, vol. 29, no. 4, pp. 293–313, jun 2004. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0306437903000723

[23] K. McGarry, "A survey of interestingness measures for knowledge discovery," *The Knowledge Engineering Review*, vol. 20, no. 01, p. 39, 2005.

[24] H. Kagdi, M. Gethers, and D. Poshyvanyk, "Integrating conceptual and logical couplings for change impact analysis in software," *Empirical Software Engineering*, vol. 18, no. 5, pp. 933–969, oct 2013. [Online]. Available: http://link.springer.com/10.1007/s10664-012-9233-9

[25] T.-d. B. Le and D. Lo, "Beyond support and confidence: Exploring interestingness measures for rule-based specification mining," in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 331–340. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7081843

[26] T. Rolfsnes, L. Moonen, S. Di Alesio, R. Behjati, and D. W. Binkley, "Improving change recommendation using aggregated association rules," in *International Conference on Mining Software Repositories (MSR)*. ACM, 2016, pp. 73–84. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2901739.2901756

[27] A. Alali, "An Empirical Characterization of Commits in Software Repositories," Ms.c, Kent State University, 2008.

[28] H. J. Hilderman, Robert and Hamilton, *Knowledge discovery and measures of interest*. Springer Science & Business Media, 2013.

[29] K. Herzig and A. Zeller, "The impact of tangled code changes," in *Working Conference on Mining Software Repositories (MSR)*. IEEE, may 2013, pp. 121–130. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6624018

[30] K. Herzig, S. Just, and A. Zeller, "The impact of tangled code changes on defect prediction models," *Empirical Software Engineering*, vol. 21, no. 2, pp. 303–336, apr 2016. [Online]. Available: http://link.springer.com/10.1007/s10664-015-9376-6

[31] O. Maimon and L. Rokach, *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Springer, 2010. [Online]. Available: http://books.google.com/books?hl=en{&}lr={&}id=S-XvEQWABeUC{&}oi=fnd{&}pg=PR21{&}dq=Data+Mining+and+knowledge+discovery+handbook{&}ots=LBVkfoBx6S{&}sig=u6c1n2kopRhLrbpg5M0FhvYhFqk{%}5Cnhttp://www.springerlink.com/index/10.1007/978-0-387-09823-4http://link.springer.com/1

[32] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2011.

[33] E. Keogh, S. Lonardi, and C. A. Ratanamahatana, "Towards parameter-free data mining," *Sigkdd*, pp. 206–215, 2004. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1014052.1014077

[34] Z. Zheng, R. Kohavi, and L. Mason, "Real world performance of association rule algorithms," in *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2001, pp. 401–406. [Online]. Available: http://portal.acm.org/citation.cfm?doid=502512.502572

[35] W. Lin, S. A. Alvarez, and C. Ruiz, "Efficient Adaptive-Support Association Rule Mining for Recommender Systems," *Data Mining and Knowledge Discovery*, vol. 6, no. 1, pp. 83–105, 2002. [Online]. Available: http://link.springer.com/10.1023/A:1013284820704

[36] N. Jiang and L. Gruenwald, “Research issues in data stream association rule mining,” *ACM SIGMOD Record*, vol. 35, no. 1, pp. 14–19, mar 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1121995.1121998

[37] A. Silva and C. Antunes, “Constrained pattern mining in the new era,” *Knowledge and Information Systems*, vol. 47, no. 3, pp. 489–516, 2016. [Online]. Available: http://link.springer.com/10.1007/s10115-015-0860-5

[38] F. Jaafar, Y.-G. Guéhéneuc, S. Hamel, and G. Antoniol, “Detecting asynchrony and dephase change patterns by mining software repositories,” *Journal of Software: Evolution and Process*, vol. 26, no. 1, pp. 77–106, jan 2014. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/smr.504/fullhttp://doi.wiley.com/10.1002/smr.1635

[39] D. M. German, “An empirical study of fine-grained software modifications,” *Empirical Software Engineering*, vol. 11, no. 3, pp. 369–393, 2006.

[40] A. Alali, H. Kagdi, and J. I. Maletic, “What’s a Typical Commit? A Characterization of Open Source Software Repositories,” in *International Conference on Program Comprehension (ICPC)*. IEEE, 2008, pp. 182–191. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4556130

[41] C. Kolassa, D. Riehle, and M. A. Salim, “The empirical commit frequency distribution of open source projects,” in *International Symposium on Open Collaboration (WikiSym)*. ACM, 2013, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2491055.2491073

Paper D

What are the Effects of History Length and Age on Mining Software Change Impact?

Leon Moonen, Thomas Rolfsnes, Stefano Di Alesio
and Dave W. Binkley

Accepted for publication in the main research track of the 16th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM), October 2-3, 2016. Further invited and submitted to the SCAM special issue published by the Springer journal Empirical Software Engineering (EMSE).

© 2016 IEEE
The layout has been revised.

Abstract

The goal of Software Change Impact Analysis is to identify artifacts (typically source-code files or individual methods therein) potentially affected by a change. Recently, there has been increased interest in mining *software change impact based on evolutionary coupling. A particularly promising approach uses association rule mining to uncover potentially affected artifacts from patterns in the system's change history. Two main considerations when using this approach are the* history length, *the number of transactions from the change history used to identify the impact of a change, and* history age, *the number of transactions that have occurred since patterns were last mined from the history. Although history length and age can significantly affect the quality of mining results, few guidelines exist on how to best select appropriate values for these two parameters.*

In this paper, we empirically investigate the effects of history length and age on the quality of change impact analysis using mined evolutionary coupling. Specifically, we report on a series of systematic experiments using three state-of-the-art mining algorithms that involve the change histories of two large industrial systems and 17 large open source systems. In these experiments, we vary the length and age of the history used to mine software change impact, and assess how this affects precision and applicability. Results from the study are used to derive practical guidelines for choosing history length and age when applying association rule mining to conduct software change impact analysis.

1 Introduction

When software systems evolve, the interactions in the source code grow in number and complexity. As a result, it becomes increasingly challenging for developers to predict the overall effect of making a change to the system. Change Impact Analysis [1] has been proposed as a solution to this problem, aimed at identifying software artifacts (e.g., files, methods, classes) affected by a given change. Traditionally, techniques for change impact analysis are based on static or dynamic analysis, which identify dependencies, for example, methods calling or called by a changed method [2–4]. However, static and dynamic analysis are generally language-specific, making them hard to apply to modern heterogeneous software systems [5]. In addition, dynamic analysis can involve considerable overhead (e.g., from code instrumentation), while static analysis tends to over-approximate the impact of a change [6].

To address these challenges, alternative techniques have been proposed that identify dependencies through *evolutionary coupling* [7–10]. In essence, evolutionary coupling exploits a developer's inherent knowledge of the dependencies in the system, which manifest themselves through commit comments, bug reports, context switches in IDEs, and so on [8]. These couplings differ from those found through static and dynamic analysis, because they are based on how the software system has evolved over time, rather than how system components are interconnected.

This paper considers *historical co-change* between artifacts as the basis for uncovering evolutionary coupling. Known techniques [10–13] for mining evolutionary couplings from artifact co-changes build on *association rule mining* (or *association rule learning*) [14], and differ in the way that the association rules are generated from the history. Nevertheless, key to all such techniques is the *history* learned from. There are two main factors related to the history that impact the mined rules: (1) the *history length*, the number of transactions in the history considered while mining co-change patterns, and (2) the *history age*, the number of transactions that have occurred since these patterns were mined. The resulting rules directly affect the quality of any change impact analysis based on mined evolutionary coupling. However, while reviewing the literature, we found that the effects of history length and age on mined association rules have not been systematically studied. We address this shortcoming.

Contributions: This paper builds upon our previous work that explored the extent to which history length and age affect the quality of software change impact analysis via association rule mining [15]. We present a series of systematic experiments using the change histories of two large industrial systems and 17 large open source systems. In particular, this paper extends our previous work in the following key respects: (1) We extend our analysis to include two more state-of-the-art software change mining techniques Co-Change and Rose in addition to the Tarmaq technique covered in our previous work. (2) We refine the co-change histories used in the analysis from a coarse-grained file-level to a more *practical fine-grained level* consisting of method-level co-change information for all parseable source-code, and file-level co-change information for un-parseable files. (3) We include an additional longitudinal study that considers change histories covering the *complete evolution history* of seven open source systems, which were selected based on the fact that their history is significantly longer (up to 540 000 transactions) than the 50000 transactions considered earlier. (4) We include a new research question that investigates the stability of recommendation quality throughout the evolution history given a particular history length and history age. (5) We strengthen the power of the statistical analysis used to address all our research questions. (6) Finally, we extend our discussion of background material and related work. We use the results from these investigations to derive practical guidelines for selecting an appropriate system-specific value for history length and for determining at what age a model has sufficiently deteriorated to benefit from rebuilding. The guidelines enable a team of engineers to best exploit association rule mining for change impact analysis.

Overview: section 2 provides background on mining evolutionary coupling.

section 3 presents our research questions. section 4 describes the setup of our empirical investigation, whose results are presented in section 5. Finally, section 6 discusses the threads to validity, section 7 presents related work, and section 8 provides some concluding remarks.

2 Mining Software Change Impact

We use historical co-change between artifacts to uncover evolutionary coupling. Such co-change data can, for example, be found as revisions in a project's version control system [16], as fixes to a bug in an issue tracking system [17], or by instrumenting the development environment [18]. Most techniques that uncover evolutionary coupling from co-change data build on *association rule mining*, an unsupervised learning technique that discovers relations between artifacts (referred to as *items* in the more general context) of a dataset [14].

Association rules are implications of the form $A \rightarrow B$, where A is referred to as the *antecedent*, B as the *consequent*, and A and B are disjoint sets. For example, consider the classic application of analyzing shopping cart data: if multiple transactions include bread and butter then a potential association rule is *bread → butter*, which can be read as *"if you buy bread, then you are likely to buy butter."*

While mining evolutionary coupling from historical co-change data, the artifacts we consider are the files and methods of a system[1] and the sequence (history) $\mathcal{T}$ of transactions, to be a list of past *commits* from a versioning system. More specifically, a transaction $T \in \mathcal{T}$ is the set of artifacts that were either changed or added while addressing a given bug fix or feature addition, hence creating a *logical dependence* between them [19].

As originally defined [14], association rule mining generates rules that express patterns in a complete data set. However, some applications can exploit a more focused set of rules. *Targeted association rule mining* [20] focuses the generation of rules by applying a constraint. An example constraint specifies that the antecedent of all mined rules belongs to a particular set of files, which effectively reduces the number of rules that need to be created. This reduction drastically improves the execution time of rule generation [20].

When performing change impact analysis, rule generation is constrained based on a *change set*, also known as a *query*. For example, the set of modified artifacts since the last commit. In this case, only rules with at least one

[1] Note that various granularity choices are possible since the algorithms are granularity agnostic; if fine-grained co-change data is available (or computable), the same algorithms will relate methods or variables just as well as more coarse-grained files. In this paper we consider a *practical* fine-grained level that uses method-level information where possible (i.e., for source files that can be parsed, as discussed later in the paper), and file-level information otherwise (e.g., documents, build files, and configuration files).

changed artifact in the antecedent are created. The resulting impacted artifacts are those found in the rule consequents. Thus, the output of change impact analysis (the *impact set*) is the set of artifacts that are historically changed alongside the elements of the change set.

Only a few targeted association rule mining algorithms have been considered in the context of change impact analysis: the Rose algorithm by Zimmerman et al. [11], the FP-tree algorithm by Ying et al. [12], the Co-Change algorithm by Kagdi et al. [13], and the Tarmaq algorithm introduced in our earlier work [10]. The algorithms differ in terms of constraints on how the query is matched against the transactions of the history. For example, given a change set $\{a, b, c\}$, Rose and FP-tree only uncover those artifacts that ever change in the history together with *all* artifacts in the query $\{a, b, c\}$. This strict matching constraint is aimed at obtaining a more precise impact set, but an analysis of the algorithm's *applicability* showed that the constraint also prevents the algorithms from produce an answer more often than not [10]. In contrast, Co-Change uncovers artifacts that ever change in the history together with *any* of a, or b, or c. This more lenient constraint is aimed at giving more answers, which are, however, potentially noisy; since the answers are only based on one matching element, they can have little relation to the full query. Finally, Tarmaq reports the artifacts that have co-changed with largest possible subset of the query, a constraint aimed at dynamically balancing the precision of a complete match with improved applicability [10]. In cases where a match of the complete query is possible, Rose, FP-tree, and Tarmaq will give the exact same result. In cases where only a subset of the query can be matched, Rose and FP-tree fail to produce an impact set, while Tarmaq provides the impact set that results from the largest possible match between the query and the change history.

3 Research Questions

It is regularly surmised in mining literature [11, 21, 22] that learning from too short or too long a history (in our case to few or two many commits) results in a suboptimal outcome, respectively because not enough knowledge about the system can be uncovered, or because outdated information introduces noise. We aim to better understand the influence of *history length* via the following research question:

RQ 1. *What influence does* history length *have on impact analysis quality?*

We refine RQ 1 using the following subquestions:

RQ 1.1. *Can we identify a* lower bound *on the history length that is needed to learn enough about the system to produce acceptable impact analysis results?*

RQ 1.2. *Do we see a* diminishing return *in impact analysis quality as history length increases?*

RQ 1.3. *Can we identify an* upper bound *on history length where outdated knowledge starts to negatively affect our analysis causing quality to decrease below acceptable levels?*

A closely related aspect is *history age*, which we define as the number of transactions that have occurred since the most recent transaction of the history used to conduct the analysis. History age basically tells us how long a model can successfully be used to make predictions regarding a system. Knowledge about the quality of impact analysis based on older histories gives valuable input regarding the feasibility of an incremental approach that reuses older association rules.

RQ 2. *What influence does* history age *have on impact analysis quality?*

We refine RQ 2 using the following subquestions:

RQ 2.1. *Can we identify an* upper bound *on the history age beyond which the generated model has grown too old and can no longer produce acceptable impact analysis results?*

RQ 2.2. *Is there a point at which impact analysis quality* ceases to deteriorate *as history age increases?*

Next, we investigate the possibility of providing project-specific advise for values of history length and history age:

RQ 3. *Can we predict good values for history length and age for a given software-system based on characteristics of its change-history (such as the average transaction size and the number of developers)?*

Finally, we investigate the sensitive of the algorithms using a common history length and history age but at different points in the system's evolution history:

RQ 4. *How does choosing a particular* history length *and* history age *affect impact analysis quality throughout the evolution history?*

Scope of investigation: To ensure a complete understanding, we will initially investigate the effects of history length and age at a *coarse* level, and progressively zoom in at *finer* levels of granularity for areas of interest indicated by the coarse study.

Moreover, based on our initial results [15] with respect to RQ 1.3, this paper includes an additional *longitudinal* study that considers the *complete evolution history* of seven open source systems. These systems were selected because their available change history is significantly longer than the histories available for other systems. This combination allows us to cover both the width of a substantial set of systems, and the depth of a long evolution history.

4 Empirical Study

We perform a comprehensive empirical study to assess the effects of history length and age on the quality of change impact analysis through mined evolutionary coupling. To consider the influence that the mining algorithm has on the study, we experiment with three of the four algorithms introduced in section 2 (we omit FP-TREE because it produces the same impact sets as ROSE). The goal of our study is to answer the research questions introduced in section 3 by controlling the history length and age while mining change impact on several large software systems.

The remainder of this section details the design of our empirical study and is organized as follows: subsection 4.1 introduces the software systems included in the study. subsection 4.2 describes the strategy we use to systematically vary history length and age. In Sections 4.3, 4.4 and 4.5 we describe how we use targeted association rule mining to generate change impact sets for a change set (i.e., a *query*) of artifacts. Finally, in subsection 4.6 we introduce the two measures used to evaluate the quality of the generated change impact sets.

4.1 Subject Systems

To assess targeted association rule mining in a variety of conditions, we selected 19 large systems having varying characteristics, such as size and frequency of transactions, number of artifacts, and number of developers. Two of these systems come from our industry partners, Cisco Norway and Kongsberg Maritime (KM). Cisco Norway is the Norwegian division of Cisco Systems, a worldwide leader in the production of networking equipment. We consider their software product line for professional video conferencing systems, developed by Cisco Norway. KM is a leader in the production of systems for positioning, surveying, navigation, and automation of merchant vessels and offshore installations. We consider a common software platform KM uses across various systems in the maritime and energy domain.

The other 17 systems, all well known open-source projects, are reported in Table D.1 along demographics illustrating their diversity. The table shows

Table D.1: Characteristics of the evaluated software systems (based on our extraction of the last 50 000 transactions for each).

Software System	History (in yrs)	Unique # files	Languages used*
CPython	12.05	7725	Python (53%), C (36%), 16 other (11%)
Mozilla Gecko	1.08	86650	C++ (37%), C (17%), JavaScript (21%), 34 other (25%)
Git	11.02	3753	C (45%), shell script (35%), Perl (9%), 14 other (11%)
Apache Hadoop	6.91	24607	Java (65%), XML (31%), 10 other (4%)
HTTPD	19.78	10019	XML (56%), C (32%), Forth (8%), 19 other (4%)
IntelliJ IDEA	2.61	62692	Java (71%), Python (17%), XML (5%), 26 other (7%)
Liferay Portal	0.87	144792	Java (71%), XML (23%), 12 other (4%)
Linux Kernel	0.77	26412	C (94%), 16 other (6%)
LLVM	4.54	25600	C++ (71%), Assembly (15%), C (10%), 16 other (6%)
MediaWiki	11.69	12252	PHP (78%), JavaScript (17%), 11 other (5%)
MySQL	10.68	42589	C++ (57%), C (18%), JavaScript (16%), 24 other (9%)
PHP	10.82	21295	C (59%), PHP (13%), XML (8%), 24 other (20%)
Ruby on Rails	11.42	10631	Ruby (98%), 6 other (2%)
RavenDB	8.59	29245	C# (52%), JavaScript (27%), XML (16%), 12 other (5%)
Subversion	14.03	6559	C (61%), Python (19%), C++ (7%), 15 other (13%)
WebKit	3.33	281898	HTML (29%), JavaScript (30%), C++ (26%), 23 other (15%)
Wine	6.6	8234	C (97%), 16 other (3%)
Cisco Norway	2.43	64974	C++, C, C#, Python, Java, XML, other build/config
Kongsberg Maritime	15.97	35111	C++, C, XML, other build/config

Software System	Unique # artifacts	Avg. # artifacts per commit	Nr. of Devs	Mode Inter-Commit	Median Inter-Commit	Avg. Commit Streak	Median Commit Streak
CPython	30090	4.52	159	0	0	5.97	4
Mozilla Gecko	231850	12.28	1047	0	11	2.66	1
Git	17716	3.13	1404	0	0	2.22	1
Apache Hadoop	272902	47.79	126	0	5	2.63	2
HTTPD	29216	6.99	119	0	1	7.85	5
IntelliJ IDEA	343613	12.6	194	0	4	2.58	1
Liferay Portal	767955	29.9	212	0	2	6.26	2
Linux Kernel	161022	5.5	3256	0	0	3.10	1
LLVM	66604	5.91	530	0	6	3.18	2
MediaWiki	12267	3.31	541	0	1	1.65	1
MySQL	136925	10.66	274	0	0	36.90	2
PHP	53510	6.74	471	0	0	3.33	2
Ruby on Rails	10631	2.56	3497	0	1	0.99	0
RavenDB	139415	18.18	259	0	0	2.84	1
Subversion	46136	6.36	91	0	1	5.95	4
WebKit	397850	18.12	393	0	12	2.68	2
Wine	126177	6.68	517	0	0	3.15	1
Cisco Norway	251321	13.62	-	-	-	-	-
Kongsberg Maritime	35111	5.08	-	-	-	-	-

* languages used by open source systems are from `http://www.openhub.net`, percentages and demographics for the industrial systems are not disclosed.

that the systems vary from medium to large size, with close to 300 000 different files for one system, nearly 768 000 artifacts in another, and almost 3 500 developers contributing to a third. For each system, we extracted the 50 000 most recent transactions (*commits*). This number of transactions covers vastly different time spans across the systems, ranging from almost 20 years in the case of HTTPD, to a little over 10 months in the case of the Linux kernel. We also report six "demographic" characteristics that we expect to be useful for answering RQ3:

1. *Number of (unique) artifacts* appearing in the history of a system;
2. *Average Commit Size:* the average number of artifacts appearing in a transaction in the history of a system;
3. *Number of Developers* who committed at least one transaction in the history of a system;
4. *Mode and Median Inter-Commit Time:* the inter-commit time is the time between two commits by the same developer, measured as a number of commits;
5. *Average and Median Commit Streaks:* a commit streak is the number of consecutive commits by the same developer in the history of a system.

Finally, the rightmost column of the upper table shows the programming languages used in each system, as an indication of heterogeneity.

4.2 History Length and Age

Given that the time span covered by 50 000 commits varies considerably across the systems in our study, we choose to express history length and age as a *number of transactions*, rather than using calendar time. This sets the same baseline for each system, enabling a meaningful comparison of the effects of history length and age across systems.

We refer to a fixed combination of history length and age as a *scenario*. In our coarse-grained study, we examine 24 scenarios pairing history lengths of 5 000, 15 000, 25 000, and 35 000 commits with ages of zero (no age), 1 000, 2 000, 3 000, 4 000 and 5 000 commits.

Preliminary results of the coarse-grained study highlighted large variations in change impact analysis quality for small length and age values, showing that the quality rapidly decreases with history aging, while increasing with longer histories. To zoom in on these areas, we conduct two additional fine-grained studies in which we respectively investigate small history lengths (for a fixed age of zero), and small ages (for a fixed history length of 35 000 commits). In each of the studies, we examine three intervals of progressively finer granularity for the variable of interest: (a) from 0 to 2 000 commits by every 100 commits; (b) from 0 to 200 commits by every 10 commits; (c) from 0 to 20 commits by every single commit. Note that we skip history lengths of zero commits because, trivially, no association rules can be

Table D.2: Characteristics of the seven software systems selected for the longitudinal study (ordered by increasing number of available transactions for each).

Software System	Available commits	History (in yrs)	Unique # of artifacts	Avg. # artifacts per commit
Wine	110950	22.81	227223	8.02
LLVM	118967	14.88	106185	5.07
IntelliJ IDEA	159020	11.47	1083032	17.23
Liferay Portal	171507	10.01	1581784	26.95
WebKit	171604	14.64	808437	17.34
Mozilla Gecko	430127	18.07	1016343	10.87
Linux Kernel	542098	10.99	953194	6.5

mined from an empty history Thus, we consider 60 scenarios for small history lengths and age zero, and 63 scenarios for small ages and a history of 35 000 commits. We refer to the fine-grained collections of scenarios characterized by each of these ranges as *lengthX* and *ageX*, where *length* and *age* specify the context where the range is used, and *X* specifies the upper bound of the range. For example, *age20* represents the fine-grained collection containing the scenarios with history length 35 000 and age in $[0, 1, 2, \ldots 20]$. To investigate fine-grained variations on a larger scale, we also consider the collection *length35k*, which varies history length from 0 to 20 in steps of 1 commit, from 20 to 200 commits in steps of 10 commits, and then from 200 to 35 000 commits in steps of 100 commits.

We do not consider a similar large interval for history age, as the preliminary coarse-grained study did not show significant variations in change impact analysis quality for age values larger than about 2 000 commits.

Finally, based on our initial findings [15], this paper adds a longitudinal study to get a more conclusive answer to RQ 1.3. In this study we analyze the *complete evolution history* of seven of our subject systems to better understand if there is an upper-bound on history length where outdated knowledge starts to negatively affect recommendation quality. Table D.2 gives an overview of the characteristics of the seven systems that were used in this investigation, which were selected because of their significantly longer evolution histories compared to other available other systems. The scenarios considered in the *longitudinal* study fix age at zero and consider history lengths from 10 000 up to the maximum available history for each system (column two of Table D.2) in steps of 10 000 commits.

4.3 History Filtering

It is a common practice in association rule mining to filter the history to remove transactions larger than a certain size [11, 12, 23, 24]. Filtering reduces noise by removing large transactions that are likely not relevant for evolutionary coupling, such as mass license updates or version bumps.

In previous work, we considered the effect of transaction filtering size on the quality of change impact analysis using association rule mining [25]. That study was conducted in a similar setting as this paper, and found that filtering transactions larger than eight items gave the best results. Therefore, for each scenario, we mine association rules from a filtered history containing transactions with at most eight items.
One challenge faced by association rule mining is that large transactions lead to a combinatorial explosion in number of association rules [14]. Fortunately, as seen in Figure D.1, which provides violin plots of transaction size for the individual systems, transaction sizes are heavily skewed towards smaller transactions.

Unfortunately, as also seen in the violin plots, there exist outlier transactions containing 10 000 or more artifacts. To combat the combinatorial explosion challenge raised by such large commits, it is common to filter the history by removing transactions larger than a certain size [11, 12, 23, 24]. Filtering reduces noise by removing large transactions that are likely not relevant for evolutionary coupling, such as mass license updates or version bumps.

In an attempt to reflect *most* change impact analysis scenarios, we employ a quite liberal filtering and remove only those transactions larger than *300* artifacts. The rational behind choosing this cutoff is that for each program at least 99% of all transactions are smaller then 300 artifacts. In most cases, the percentage is well above 99% of the available data.

4.4 Query Generation and Execution

Conceptually, a *query* Q represents a set of artifacts that a developer changed since the last synchronization with the version control system. Recall that the main assumption behind evolutionary coupling is that artifacts that frequently change together are likely to depend on each other. The key idea behind our evaluation is to sample a transaction T from the history, and randomly partition it into a non-empty query Q and a non-empty *expected outcome* $E \overset{\text{def}}{=} T \setminus Q$. This allows us to evaluate to what extent our change impact analysis technique is able to estimate E from Q for a given scenario.

From each filtered history we take a sample of 1100 recent transactions,[2]

[2]For a normally distributed population of 50 000, a minimum of 657 samples is required to attain 99% confidence with a 5% confidence interval that the sampled transactions are representative of the population. Since we do not know the distribution of transactions, we correct the

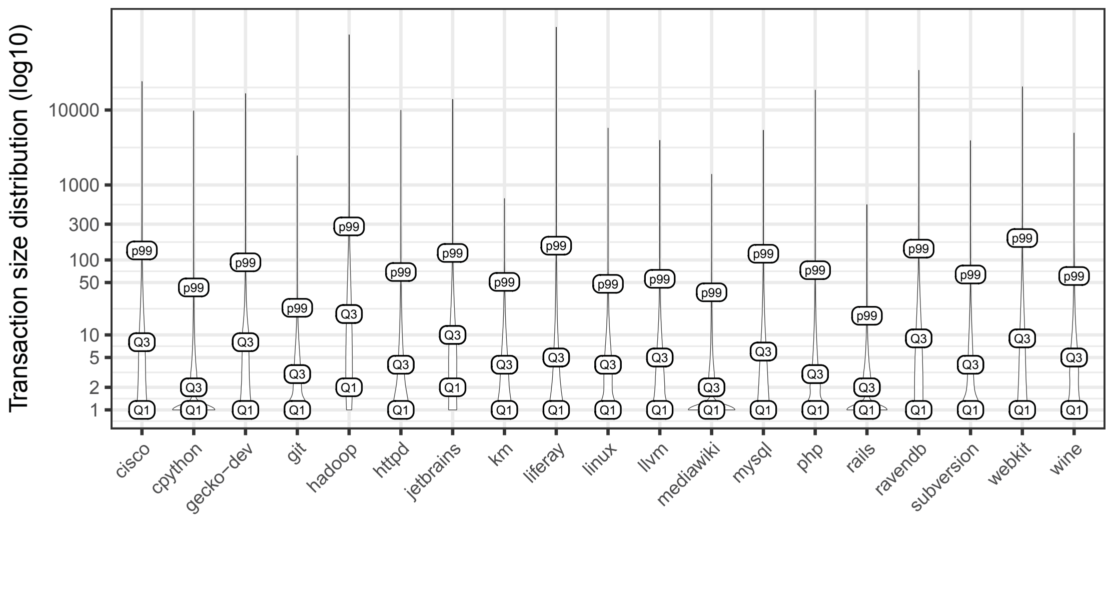


Fig. D.1: An overview of the distributions of transactions sizes for each subject system.

with the constraint that the transaction must contain at least two artifacts. This constraint ensures that, at the minimum, a transaction can be split into a query of at least one artifact and an expected outcome of at least one. Each of these transactions is randomly split into a non-empty query and a non-empty expected outcome. The resulting 1100 queries are executed using each of the three algorithms, and for each of (a) the 24 scenarios in the coarse-grained study, (b) the 123 scenarios in the fine-grained studies, and (c) the 385 scenarios in the *length35k* study. This setup yields a total of $1100 \cdot 3(24 + 123 + 386) = 1\,758\,900$ data points for each of the 19 systems, where each data point is the estimated impact set for a given query (33.42 million data points in total). The longitudinal study adds another 151 scenarios for the seven systems considered, yielding an additional $1100 \cdot 3 \cdot 151 = 498\,300$ data points which brings the total close to 34 million. Note that in each of the scenarios, we only mine from transactions that are *older* than the transaction T that was used to generate the query. In this was we respect the historical time-line for the query and the transactions used to address that query.

4.5 Estimating the Impact of a Change

All queries are executed using each of the three rule mining algorithm. Recall from section 2 that, in the context of targeted association rule mining, execut-

sample size to the number needed for a non-parametric test to have the same ability to reject the null hypothesis. This correction is done using the Asymptotic Relative Efficiency (ARE). As AREs differ for various non-parametric tests, we choose the lowest coefficient, 0.637, yielding a conservative minimum sample size of $657/0.637 = 1032$ transactions. Hence, a sample size of 1100 is more than sufficient to attain 99% confidence with a 5% confidence interval that the samples are representative of the population.

ing a query Q entails the generation of a set of association rules. The *impact set* of Q is the list of consequents of the rules generated for Q, where such rules are ranked according to their *interestingness*. While a number of interestingness measures have been defined over the years, in our study we rank association rules based on *support* and *confidence* [14]. The support of a rule is the percentage of transactions in the history containing both the antecedent and the consequent of a rule. Intuitively, high support suggests that a rule is more likely to hold because there is more historical evidence for it. On the other hand, the confidence of a rule is the number of historical transactions containing both the antecedent and the consequent divided by the number of transactions that contain only the antecedent. Intuitively, the higher the confidence, the higher the chance that when items in the antecedent of a rule change, the items in the consequent also change. We configure each mining algorithm to rank rules using support, breaking ties based on confidence. This strategy has been applied in several association rule mining approaches for software change impact analysis [11, 12, 23, 24]. Note that we consider only the largest interestingness score for each consequent. This means that, for the purpose of this study, we do not consider rule aggregation strategies [26].

4.6 Quality Measures

We empirically assess the quality of the change impact sets generated using two measures, Average Precision (AP) and Applicability.

Definition 2 (Average Precision). *Given a query Q, its impact set I_Q, and expected outcome E_Q, the average precision AP of I_Q is given by*

$$AP(I_Q) \overset{def}{=} \sum_{k=1}^{|I_Q|} P(k) * \triangle r(k) \tag{D.1}$$

where $P(k)$ is the precision *calculated on the first k items in the list (i.e., the fraction of correct artifacts in the top k artifacts), and $\triangle r(k)$ is the* change in recall *calculated only on the $k-1^{th}$ and k^{th} artifacts (i.e., the number of additional correct items predicted compared to the previous rank) [27].*

As an overall performance measure for a scenario (i.e., for a given history length and age) across a system, we use the Mean Average Precision (MAP) computed over all the queries executed for a given scenario.

Average Precision is a standard measure commonly used in Information Retrieval to assess the extent to which a list of retrieved documents includes the relevant documents for a query. However, when using association rule mining, it is not always possible to generate such list. This can happen for example when there are no transactions in the history whose items changed at least once with an item in the query. While this scenario is unlikely for long histories, the chance of finding a previous transaction involving an artifact from the query decreases as the history length shortens. Therefore, we define *Applicability* as the percentage of queries for which an impact set can be generated (i.e., where the history contains transactions involving items from the query).

4.7 Bootstrapping Procedure

The distribution of AP values is unsurprisingly highly left skewed because these values drop quickly when there is no correct artifact in the first few positions. For example, consider three ranked lists for a query whose expected outcome includes a single artifact. In the first list the correct artifact is first, in the second it is second, and in the third it is third. In this case the AP values are 1.00, 0.50 and 0.33 respectively. AP values can drop even faster if there is more than one relevant artifact.

The nature of our study also requires looking at various 2-way interactions, which is infeasible for certain interactions when using non-parametric methods. For these reasons we apply bootstrapping to approximate the sampling distribution. Doing so preserves centrality (i.e., the mean does not change), but yields an approximately normal distribution of AP values.

5 Results and Discussion

This section presents the results of the coarse-grained and fine-grained analysis of history length and age described in section 4. In particular, the results of the coarse-grained study that is described first motivates two fine-grained studies: (1) the impact of various history lengths at the fixed age zero, and (2) the impact of various history ages for the fixed history length 35 000.

5.1 Coarse-grained Study

Table D.3 presents the results of an ANOVA explaining the AP values using the data of the coarse-grained study. In addition to history length and age, which are the main variables of interest, we include subject system and algorithm as as explanatory variable to allow the statistical model to account for system or algorithm specific variations. We also include all two-way interaction terms to account for possible interaction effects.

With extremely small p-values and F-Values substantially larger than one, all four primary explanatory variables are highly statistically significant. This is also reflected by the effect sizes (shown here as Partial eta^2). A QQ-plot of the residuals (not included) showed minimal deviation from the diagonal, indicating a near normal distribution.

Table D.3: ANOVA results for the coarse-grained study

Explanatory variable	Partial etaaˆ2	Sum Sq	Df	F-value	*p*-value
subject system	0.76	2973.47	18	206196.51	< 0.00001
algorithm	0.36	536.86	2	335061.64	< 0.00001
history length	0.20	236.63	3	98453.51	< 0.00001
age	0.63	1574.33	5	393022.32	< 0.00001
subject system:algorithm	0.05	49.12	36	1703.01	< 0.00001
subject system:history length	0.06	58.89	43	1709.41	< 0.00001
subject system:age	0.18	204.58	90	2837.39	< 0.00001
algorithm : history length	0.00	3.63	6	755.55	< 0.00001
algorithm : age	0.01	9.79	10	1221.59	< 0.00001
history length : age	0.01	5.80	15	482.59	< 0.00001

The interactions between the primary explanatory variables have in general a weak impact, as shown their relatively small F-value and effect size. In particular, there is only a very small interaction between the main variables of interest, history length and age, which is visible as the different slopes of the lines in the interaction plot shown in Figure D.2. This graph also shows that the MAP values for age zero are considerably higher than those of the other ages, which are bunched relatively close together. The gap going from age zero to age 1000 is the motivation for the fine-grained study zooming in on the smaller ages.

One interaction that is a bit larger is the one between subject system and age, indicating that the impact of history age on recommendation quality is to some extent system dependent, in contrast to, for example, the impact of history length or algorithm which is more or less similar over all system.

Table D.4 reports Tukey's Honest Significant Difference (HSD) test for the ANOVA of Table D.3 applied to history length and age. Tukey's test partitions values of history length and age in groups in such a way that values belonging to the same group do not yield statistically significantly

Table D.4: Tukey's HSD for History Length and History Age (each sorted on decreasing MAP values).

History Length		
length	MAP	group
35000	0.1567	a
25000	0.1530	b
15000	0.1423	c
5000	0.1155	d

History Age		
age	MAP	group
0	0.2185	a
1000	0.1430	b
2000	0.1312	c
3000	0.1225	d
4000	0.1155	e
5000	0.1096	f

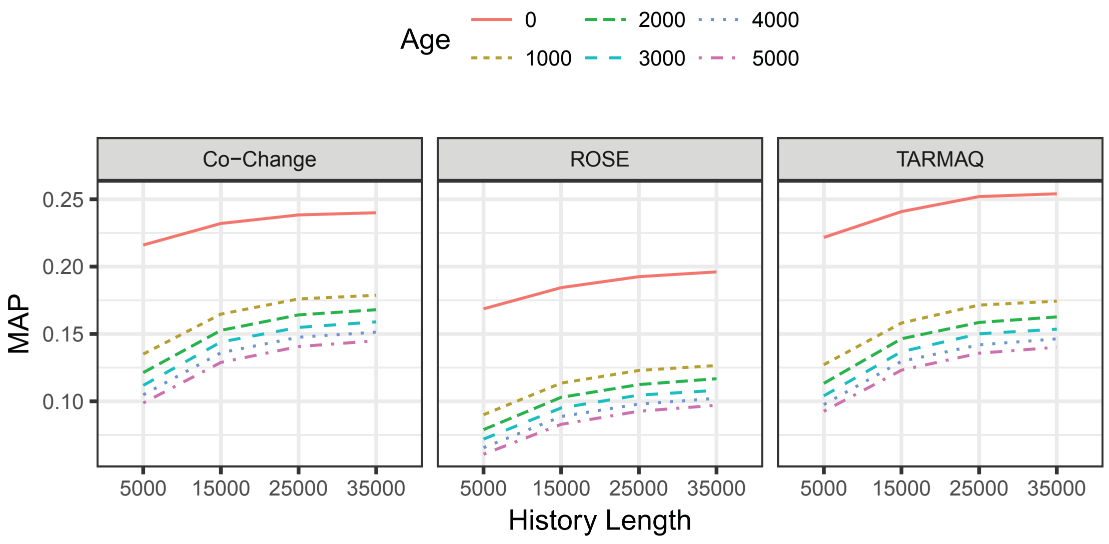


Fig. D.2: Interaction plots of Age by History Length for the various algorithms.

different MAP values. The test suggests three main conclusions: First, there are significant differences between *all* levels of each variable. Second, for history length, the best performance is attained by the largest length value of 35 000. We will use this particular length value later in the fine-grained study of history age (subsubsection 5.2). Finally, for history age, the best performance is attained by age zero, which we hence use in the fine-grained study of history length (subsubsection 5.2). Both the graphs and Tukey's HSD indicate that *very recent commits have a strong influence on the ability to predict change impacts*. These three findings motivated our fine-grained study of small history length and age.

Focusing on the age zero data only, Figure D.3 shows the applicability of TARMAQ, along with the overall MAP and the MAP for applicable queries. The applicability of TARMAQfollows the expected trend: the algorithm grows more applicable as the history length increases. The results of the coarse-grained study also suggest that the overall MAP and the MAP for applicable queries are very similar. However, intuition suggests that the difference between the two will increase as the history length shortens.

5.2 Fine-grained Study

The coarse-grained analysis motivates the study of small history lengths and ages. The results of these studies are presented and discussed in the following two subsections.

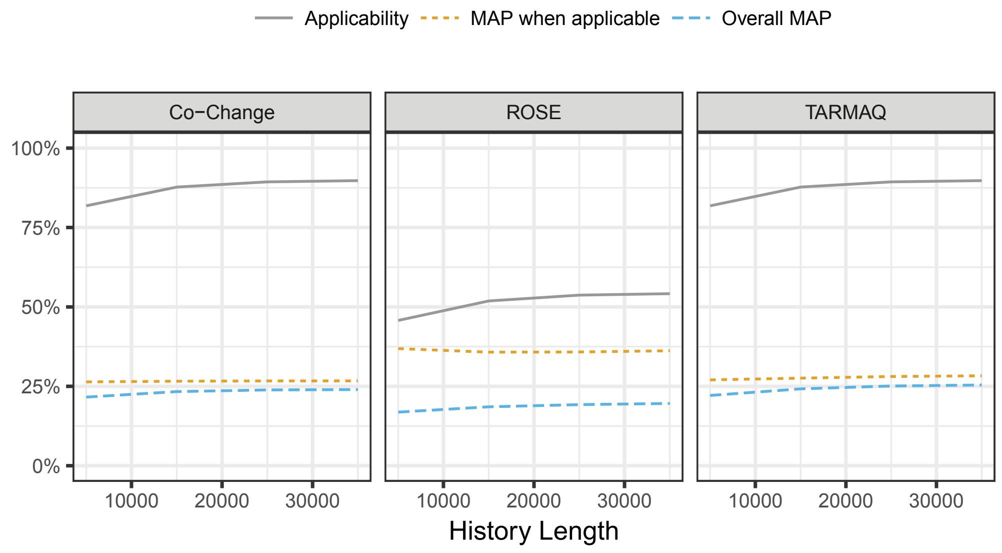


Fig. D.3: Coarse-level study of history length

History Length

Note that one challenge that shorter history lengths bring is a higher likelihood that for a given query no other commit from the history includes *any* artifacts from the query. In such cases TARMAQor CO-CHANGEare *not applicable* as they are unable to generate an impact set. Note that this occurs already at an earlier stage for ROSE, as it needs a full match of the query in the history. While it is possible to assign an AP of zero to such cases, doing so is *harsh* because the algorithm can correctly inform the user that it is not applicable. From a user perspective, this is substantially better than an incorrect impact set (where AP is truly zero). To account for this, we report three things: *applicability*, *MAP for applicable queries*, the value of MAP for only applicable queries, and *overall MAP*, the value of MAP computed using all queries including those to which an algorithm is not applicable.

An ANOVA for the fine-grained study of history lengths using age zero finds largely the same results as the coarse-grained analysis, as shown in Table D.5. Now the effects of age are factored out, the effects of history size become more prominent, and from the interactions we also see that the differences between individual systems have a greater impact on the effects of history length on recommendation quality.

Tables D.6 and D.7 show the results of Tukey's HSD for the scenario collections *length2000*, *length200*, and *length20* on respectively overall MAP and applicable MAP. The results show that for overall MAP, statistically significantly higher MAP values are produced by long histories, whereas for applicable MAP this is reversed: statistically significantly higher MAP values are produced by short histories. Note that because each column represents

Table D.5: ANOVA results for the fine-grained study of *length35k* (with *age = 0*).

Explanatory variable	Partial eta^2	Sum Sq	Df	F value	*p*-value
Subject System	0.68	5641.18	18	418709.25	< 0.0001
Algorithm	0.24	854.90	2	571083.84	< 0.0001
History Length	0.72	6879.40	55	167110.18	< 0.0001
Subject System:Algorithm	0.04	113.01	36	4194.01	< 0.0001
Subject System:History Length	0.31	1212.13	990	1635.79	< 0.0001
Algorithm:History Length	0.03	93.80	110	1139.22	< 0.0001

a separate sample of the commits, the MAP values should not be directly compared between columns, only the trends are comparable.

In particular, in the first column the scenario with a history length of 100 yields statistically significantly higher MAP than all the other scenarios. The same trend is seen in the second column, where longer history yields a strictly decreasing MAP value. Finally, the third column shows that histories of lengths as short as 1 or 2 commits are very effective in estimating the change impact set, while longer histories up to 20 commits are progressively less effective.

This data suggests that very short histories yield the best results and furthermore that the extent to which artifacts contained in past transactions are related to the artifacts in a query progressively decreases as history length increases. In other words older transactions are more likely to contain artifacts unrelated to a query. However, an explanation for this seemingly odd behavior is found in the considerably lower applicability of TARMAQas the history length shortens.

This data suggests two contrasting trends: One the one hand it shows that the longest histories yield the best overall MAP results and the extent to which artifacts contained in past transactions are related to the artifacts in a query progressively increases as history length increases. On the other hand it shows that very short histories yield the best results for applicable MAP and the extent to which artifacts contained in past transactions are related to the artifacts in a query progressively decreases as history length increases. An explanation for these contrasting trends is found in the considerably lower applicability of the algorithms as the history length shortens. This trade-off can be seen in Figure D.4, which reports the applicability, MAP, and MAP for applicable queries in the three fine-grained studies on the history length. In particular, across all granularities, applicability and MAP show an increasing trend, while the MAP for applicable queries shows a decreasing trend. This is expected because, the longer the history, the higher the chance that at least one past transaction contains artifacts related to the query, which raises applicability and (overall) MAP.

These trends continue for longer history lengths, as is evidenced by the

Table D.6: Tukey's HSD for overall MAP achieved on the *length2000*, *length200*, and *length20* collections (each sorted by decreasing MAP values).

length2000			*length200*			*length20*		
length	MAP	group	length	MAP	group	length	MAP	group
2000	0.1857	a	200	0.1329	a	20	0.0829	a
1900	0.1844	b	190	0.1319	b	19	0.0818	b
1800	0.1832	c	180	0.1314	b	18	0.0805	c
1700	0.1814	d	170	0.1298	c	17	0.0798	d
1600	0.1799	e	160	0.1289	d	16	0.0783	e
1500	0.1786	f	150	0.1271	e	15	0.0769	f
1400	0.1773	g	140	0.1261	f	14	0.0756	g
1300	0.1751	h	130	0.1241	g	13	0.0742	h
1200	0.1737	i	120	0.1224	h	12	0.0727	i
1100	0.1719	j	110	0.1200	i	11	0.0708	j
1000	0.1691	k	100	0.1181	j	10	0.0692	k
900	0.1670	l	90	0.1155	k	9	0.0675	l
800	0.1648	m	80	0.1124	l	8	0.0657	m
700	0.1614	n	70	0.1097	m	7	0.0635	n
600	0.1583	o	60	0.1059	n	6	0.0609	o
500	0.1538	p	50	0.1019	o	5	0.0576	p
400	0.1486	q	40	0.0968	p	4	0.0539	q
300	0.1422	r	30	0.0913	q	3	0.0493	r
200	0.1343	s	20	0.0834	r	2	0.0427	s
100	0.1185	t	10	0.0696	s	1	0.0321	t

collection *length35k*, which is shown in Figure D.5. The analysis of this figure, combined with the results of Tukey's HSD (not shown), allow us to answer RQ1 as follows.

RQ 1. *What influence does* history length *have on impact analysis quality?*

RQ 1.1. *Can we identify a* lower bound *on the history length that is needed to learn enough about the system to produce acceptable impact analysis results?*

Given the leveling off of applicability as history length grows, our analysis suggests that 25 000 commits is the point at which there is sufficient history to learn enough about the system to produce acceptable impact analysis results. Of course those willing to tolerate lower applicability, could consider shorter histories.

RQ 1.2. *Do we see a* diminishing return *in impact analysis quality as history length increases?*

In short, Yes. Our analysis for the three algorithms shows that their performance consistently increases for histories up to 15 000 commits, and then levels off and remains stable for longer histories (Figure D.5). Therefore, we identify the point of diminishing return as 15 000 commits.

Table D.7: Tukey's HSD for applicable MAP achieved on the *length2000*, *length200*, and *length20* collections (each sorted by decreasing MAP values).

length2000			*length200*			*length20*		
length	MAP	group	length	MAP	group	length	MAP	group
100	0.3383	a	10	0.3765	a	1	0.4017	a
200	0.3298	b	20	0.3667	b	2	0.3989	b
300	0.3224	c	30	0.3602	c	3	0.3949	c
400	0.3196	d	40	0.3513	d	4	0.3929	d
500	0.3165	e	50	0.3488	e	5	0.3886	e
600	0.3138	f	60	0.3449	f	6	0.3870	f
700	0.3118	g	70	0.3426	g	7	0.3850	g
800	0.3106	h	80	0.3397	h	8	0.3832	h
900	0.3086	i	90	0.3385	i	9	0.3801	i
1000	0.3078	j	100	0.3369	j	11	0.3788	j
1100	0.3074	j	110	0.3350	k	10	0.3787	j
1200	0.3060	k	120	0.3335	l	12	0.3780	j
1400	0.3058	kl	130	0.3321	m	13	0.3758	k
1300	0.3051	lm	140	0.3312	n	14	0.3749	k
1500	0.3047	mn	150	0.3299	o	15	0.3740	l
1800	0.3045	mn	160	0.3292	o	17	0.3733	lm
1600	0.3044	mno	170	0.3277	p	16	0.3730	m
1700	0.3043	no	180	0.3272	pq	18	0.3718	n
1900	0.3041	no	190	0.3265	q	19	0.3716	no
2000	0.3037	o	200	0.3254	r	20	0.3709	o

RQ 1.3. *Can we identify an* upper bound *on history length where outdated knowledge starts to negatively affect our analysis causing quality to decrease below acceptable levels?*

No, our analysis of histories up to 35 000 transactions does not show any evidence of performance degrading because of older outdated commits. In subsection 5.4 we will consider histories longer than 35 000 transactions.

History Age

Parallel to Table D.3, the ANOVA for the three age collections (not shown), finds age and subject system to be highly statistically significant. Figure D.6 show the results for Tukey's HSD on the collections *age2000*, *age200*, and *age20* for respectively overall and applicable MAP. In both cases, the scenarios all appear in age order, showing a few overlapping groups. for both MAPs and in all three collections, the scenario with age zero performs statistically better than the next smallest age. While the gap in MAP gets smaller as the ages in the scenarios get closer, the MAP differences are significant even when going from an age of one commit to an age of two commits in the *age20* collections.

Figure D.6 shows the trends for applicability, overall MAP, and MAP for applicable queries for the three algorithms. Across the collections, the drop

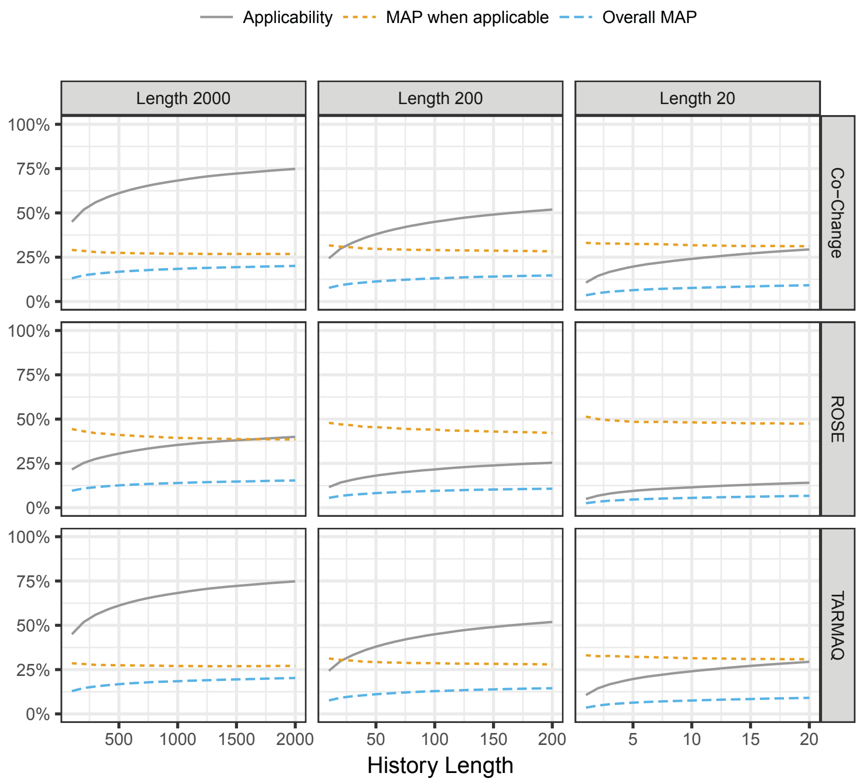


Fig. D.4: The fine-grained study's three history-length scenario collections showing the inverse relation between MAP and applicability.

off from the scenario characterized by age zero to the next age is visually evident, although it clearly gets less prominent from *age2000* to *age200*, and finally to *age20*. Moreover, we see that even though individually values differ, the trends are very similar across the three algorithms. Based on the fast deterioration of recommendation quality when age increases, we do not consider the study of larger ages than 2000 to be relevant (i.e., we do not include a study comparable to *length35k*).

Similar to RQ1, the plots in Figure D.6 and Tukey's HSD in Figure D.6 enable us to answer RQ2.

RQ 2. *What influence does* history age *have on impact analysis quality?*

RQ 2.1. *Can we identify an* upper bound *on the history age beyond which the generated model has grown too old and can no longer produce acceptable impact analysis results?*

Such a bound is a function of ones tolerance for lost precision, which depends on the use and application of targeted association rule mining for change impact analysis. Similar to the length analysis (subsubsection 5.2), the falloff in precision tends to gradually narrow. However, the falloff is initially quite steep. For example, in Table D.9 the second entry shows a reduction of 11.5%

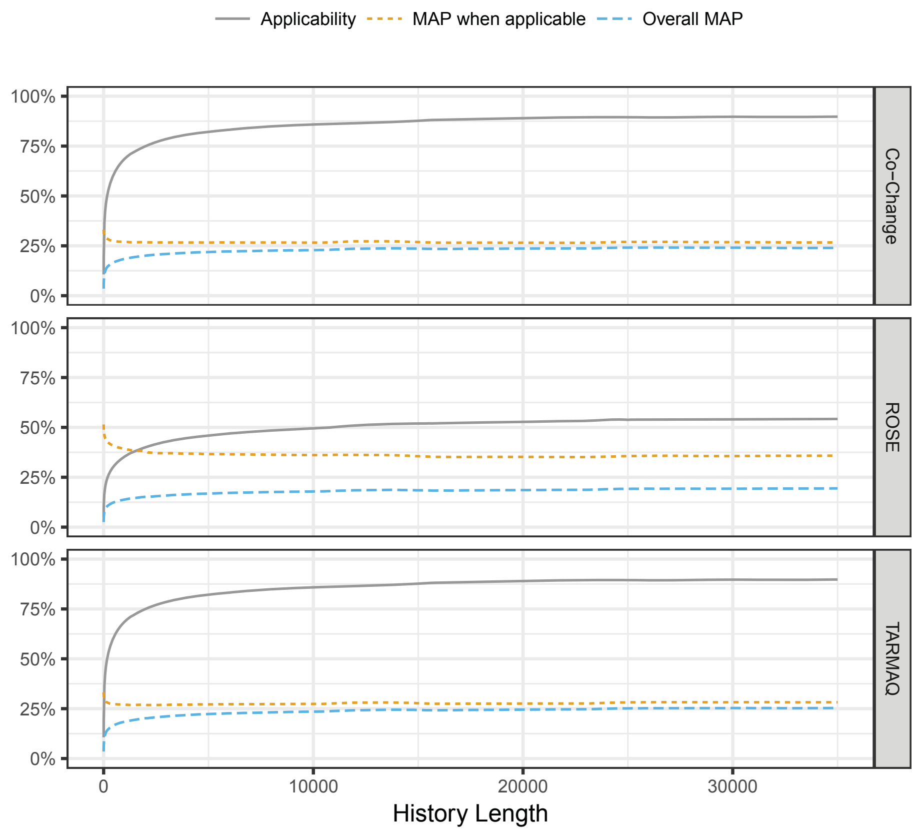


Fig. D.5: Results from the large-scale fine-grained investigation of *length35k*.

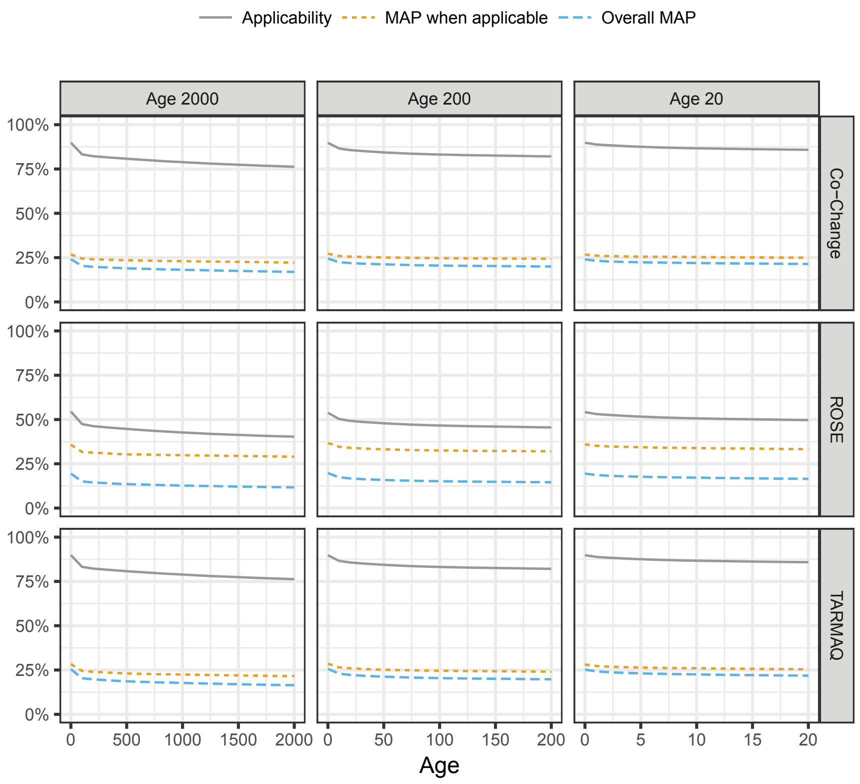


Fig. D.6: The fine-grained study's three history-age ranges shown from coarsest to finest

Table D.8: Tukey's HSD for overall MAP achieved on the *age2000*, *age200*, and *age20* collections (each sorted by decreasing MAP values).

age2000			*age200*			*age20*		
age	MAP	group	age	MAP	group	age	MAP	group
0	0.2301	a	0	0.2330	a	0	0.2290	a
100	0.1857	b	10	0.2093	b	1	0.2204	b
200	0.1798	c	20	0.2029	c	2	0.2168	c
300	0.1769	d	30	0.1994	d	3	0.2140	d
400	0.1733	e	40	0.1969	e	4	0.2124	e
500	0.1703	f	50	0.1946	f	5	0.2107	f
600	0.1686	g	60	0.1927	g	6	0.2092	g
700	0.1668	h	70	0.1907	h	7	0.2082	h
800	0.1647	i	80	0.1894	i	8	0.2074	i
900	0.1636	j	90	0.1880	j	9	0.2065	j
1000	0.1619	k	100	0.1875	j	10	0.2053	k
1100	0.1605	l	110	0.1865	k	11	0.2047	k
1200	0.1591	m	120	0.1856	l	12	0.2034	l
1300	0.1580	n	130	0.1850	lm	13	0.2033	lm
1400	0.1565	o	140	0.1844	mn	14	0.2026	mn
1500	0.1554	p	150	0.1838	no	15	0.2021	no
1600	0.1542	q	160	0.1832	op	16	0.2014	o
1700	0.1531	r	170	0.1828	p	17	0.2004	p
1800	0.1521	s	190	0.1817	q	18	0.2000	pq
1900	0.1511	t	180	0.1817	q	19	0.1995	qr
2000	0.1503	u	200	0.1810	q	20	0.1989	r

Table D.9: Tukey's HSD for applicable MAP achieved on the *age2000*, *age200*, and *age20* collections (each sorted by decreasing MAP values).

age2000			*age2000*			*age2000*		
age	MAP	group	age	MAP	group	age	MAP	group
0	0.2994	a	0	0.3052	a	0	0.2990	a
100	0.2650	b	10	0.2857	b	1	0.2917	b
200	0.2607	c	20	0.2805	c	2	0.2882	c
300	0.2582	d	30	0.2779	d	3	0.2862	d
400	0.2548	e	40	0.2758	e	4	0.2849	e
500	0.2525	f	50	0.2743	f	5	0.2836	f
600	0.2516	g	60	0.2728	g	6	0.2823	g
700	0.2509	g	70	0.2715	h	7	0.2816	gh
800	0.2488	h	80	0.2703	i	8	0.2809	hi
900	0.2485	h	90	0.2699	i	9	0.2801	ij
1000	0.2474	i	100	0.2687	j	10	0.2794	j
1100	0.2467	ij	110	0.2679	jk	11	0.2794	j
1200	0.2462	jk	120	0.2676	k	12	0.2780	k
1300	0.2456	k	130	0.2666	l	13	0.2776	kl
1400	0.2438	l	140	0.2664	l	14	0.2774	kl
1500	0.2438	l	160	0.2662	l	15	0.2771	lm
1600	0.2425	m	170	0.2659	l	16	0.2763	m
1700	0.2423	m	150	0.2659	l	17	0.2754	n
1800	0.2407	n	180	0.2647	m	18	0.2754	n
1900	0.2400	no	190	0.2644	m	19	0.2750	no
2000	0.2398	o	200	0.2641	m	20	0.2744	o

for *age2000*, 6.4% for *age200* and 2.4% for *age20*. When using a 10% tolerance cutoff of the maximum achievable applicable MAP as an arbitrary maximal acceptable loss, the upper bound for history age is 40 commits. In summary, we conclude there is a bound on age for RQ 2.1, where the actual value for this bound is a function of the user's tolerance and experience.

RQ 2.2. *Is there a point at which impact analysis quality* ceases to deteriorate *as history age increases?*

Similar to RQ 2.1, the point of diminishing deterioration is subjective, as it depends on the cutoff for the MAP values. Following RQ 2.1, we again use a 10% cutoff. The lowest applicable MAP in Table D.9 is 0.2398, which is for age 2000. Using this value, the target applicable MAP value is 0.2398 + 10% = .22638, which is crossed between age 100 and age 200. Thus, while the performance is monotonically decreasing as age increases, it does reach a point after which the remaining deterioration is not significant. Therefore, it is possible to find a point beyond which impact analysis quality ceases to deteriorate significantly as history age increases.

5.3 Project Characteristics

RQ 3. *Can we predict good values for history length and age for a given software-system based on characteristics of its change-history (such as the average transaction size and the number of developers)?*

RQ3 aims to support a team of developers working on a specific system by providing practical guidelines for selecting an appropriate value for history length and for predicting at what age a model has sufficiently deteriorated to need rebuilding.To answer this question, we build six separate linear regression models, three that predict the value of history length for Tarmaq, Co-Change, and Rose, and three that consider age. In both cases, the set of explanatory variables includes the following system demographics (see Table D.1) that aim to capture aspects of the development history, team, and development process: (1) *number of unique artifacts* in the change history, (2) *average number of artifacts in a commit*, (3) *number of developers* throughout the change history, (4) *mode and median inter-commit time* between two commits by the same developer, (5) *average and median length of commit streak* (a streak is a number of consecutive commits by the same developer). The values of these demographics were obtained by analyzing the change histories for the various systems.

In the analysis of this section, we omit Cisco and KM for which we cannot disclose the demographics. Also, similar to the analysis in subsection 5.1, the regression analysis for history length is performed using an age of zero, while the regression analysis for age is performed using a history length of 25 000.

We begin with the three linear regression models for history length. Each requires determining a target history length for each system. A simple selection would be the history length that produced the highest MAP value. Unfortunately, from the statistical analysis this value is not unique. Thus, from the set of top-performing history lengths, we selected the smallest value under the assumption that requiring less history is preferable (e.g., leads to more efficient computation of recommendations). In summary the target value for history length for each system is determined by applying Tukey's HSD test and then selecting from the top group (the 'a' group) the smallest history length.

We then fit a linear model to the data using R's *lm* function starting with all the explanatory variables and then applying backward elimination. The elimination phase removes the least statistically significant variable and then regenerates a new model. For example, in TARMAQ's initial model the variable *median inter-commit streak* has the highest p-value of 0.74. The elimination step removes this variable and then rebuilds the model. This process is repeated until only significant variables remain.

It is also possible to consider interactions between the explanatory variables. Preliminary work with the demographics made it quite evident that there were interactions among the explanatory variables. Unfortunately, with our 17 systems there is insufficient data to build a model that includes all pair-wise interactions. As a compromise an initial model was generated without any interactions and then interactions were added for all variables having a *p*-value less then 0.33. The elimination process is then applied to produce the final model. Note that in order to maintain a well-formed model some variables with *p*-values greater than 0.05 are retained in the final model when they are part of a significant interaction.

The remainder of this investigation first considers TARMAQ before turning to CO-CHANGE and then finally ROSE. The final model for TARMAQ, shown in Table D.10, is statistically significant having a *p*-value of 0.000934. The model includes three significant explanatory variables and three significant interactions, which makes it challenging to understand the effects of the variables. One standard approach to doing so looks at the pair-wise interactions when the third variable takes its mean value. The resulting interaction graphs, shown in Figure D.7, were generated from the model shown in in Table D.10 using the following values:

Variable	First Quartile	Mean	Third Quartile
Average Number of Artifacts per Commit (ANAC)	5.5	11.8	12.6
Median Inter Commit Time (MICT)	0.0	2.6	4.0
Number of Unique Artifacts (NUA, in 1000's)	30.0	167.0	232.0

Table D.10: Regression model predicting history length for TARMAQ

Explanatory variable	Estimate	Std. Error	*t*-value	*p*-value
(Intercept)	8859.2	2356.20	3.76	0.00372
Median Inter Commit Time (MICT)	1342.3	740.50	1.81	0.09995
Number of Unique Artifacts (NUA, in 1000's)	141.5	23.80	5.93	0.00014
Average Number of Artifacts per Commit (ANAC)	-673.8	342.20	-1.97	0.07769
MICT : NUA	-14.9	3.67	-4.21	0.00180
MICT : ANAC	192.3	75.60	2.54	0.02925
NUA : ANAC	-3.1	0.67	-4.59	0.00099

In addition to providing an affirmative answer to the first half of RQ3, it is interesting to consider the parameter estimates found in the model and discuss possible explanations for the relations between the explanatory variables and the response variable (i.e., history length). Starting with the three explanatory variables, the analysis shows that as median inter-commit time (MICT) increases a longer history length is needed. A potential explanation of this relation is working context: it is likely that a developer works on a sequence of related modifications, i.e. a sequence of commits from that developer likely contains related artifacts. When MICT is zero, the commits of a developer tend to follow each other directly in the history and short history lengths suffice to achieve high precision recommendations. As the value of MICT grows there are an increasing number of interleaved commits from other developers, with other working contexts, which means that longer history lengths are needed to include the relevant artifacts and achieve high precision recommendations.

The model also shows that as the number of unique artifacts (NUA) increases, so does the predicted history length. Indeed, the more artifacts contained in a system, the further apart commits containing artifacts relevant to a query are likely to be, and hence a larger history is required.

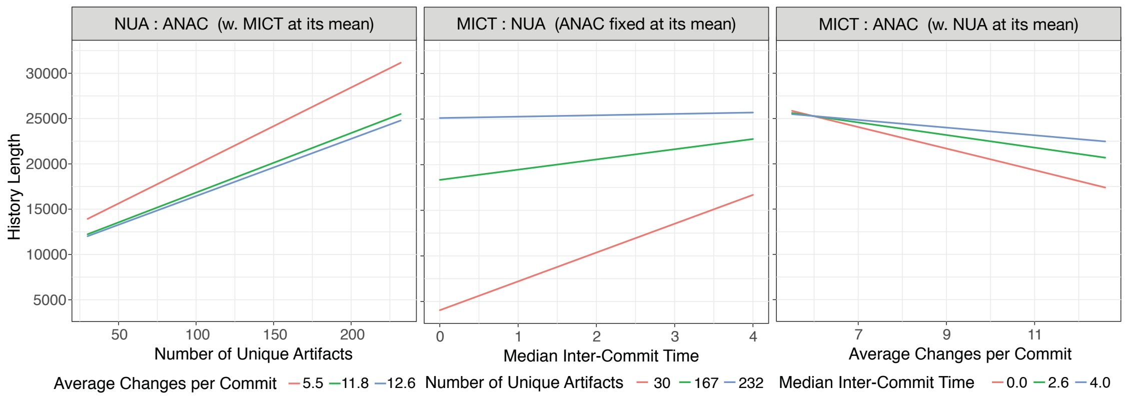


Fig. D.7: Interaction plots for the three interaction terms of the regression model shown in Table D.10

Finally, the coefficient associated with average number of artifacts per commit, is negative, thus indicating an inverse relationship. In other words, as the average "size" of a commit increases, the required history length decreases. One explanation for this relation is that larger commits contain more information per commit. For example, to conclude that changing a impacts b and c can be derived from the single size-three commit $\{a, b, c\}$, but requires two size-two commits: $\{a, b\}$ and $\{a, c\}$. Another explanation is the combination of working context and commit practices. Developers that tends to create large commits are likely to commit on a task-by-task basis, creating a transaction that contains all artifacts related to a modification task at once. This behavior makes it less likely that there is information with respect to related artifacts contained in a short history length. Conversely, developers that split a task in several smaller commits, i.e., decrease the number of artifacts per commit, spread out artifacts related to their working context over a larger number of commits, thereby increasing the history length required to achieve high precision recommendations.

Figure D.7 plots the three interactions. In the first of the three graphs, as the average number of changes per commit increases the impact of the number of unique artifacts decreases. The relative slopes of the lines indicates that this effect is not large, which is also evident from its small parameter estimate (see the last line of Table D.10). Looking at the middle graph, the relative impact of the number of unique artifacts is more pronounced as evidenced by the greater difference in the line's slopes. In this case it is interested that as this value increases, the impact of the median inter-commit time approaches zero. Conversely, for smaller systems the median inter-commit time has a greater influence. Finally, from the third chart, the average number of changes per commit has a greater impact for smaller median inter-commit times. While a higher average number of changes per commit brings a need for less history the reduction is greater them the median inter-commit time is small. The smallest history requirement comes from larger commits without intervening commits of other developers.

While statistically significant with a p-value of 0.0345, the model for Co-Change omits the median inter-commit time. Perhaps because of it's simpler rules. It also includes smaller parameter estimates (for example the coefficient for the number of unique artifacts is 141.5 with Tarmaq, but only 60.7 for Co-Change. Like the Tarmaq model the sign of the average changes per commit and the interaction of the two is negative but again both have smaller magnitude (-6.6 and -1.7, respectively). Thus overall, the influences of the explanatory variables in the Co-Change model are muted relative to the Tarmaq model.

Finally, the Rose model, which has a p-value of 0.00114, is similar to the Tarmaq model in that it includes the same explanatory variables and interactions. Furthermore, the coefficients are very similar (for example, consider the average number of changes per commit, which is -673.8 in the Tarmaq model and -673.3 in the Rose model. The largest difference is the coefficient of median inter-commit time, which in the Rose model is 1708 compared to 1342 in the Tarmaq model. This difference indicates that Rose is more susceptible to changes in the median inter-commit time than Tarmaq. Where Rose needs less history when commits are clumped by relevance but as the commits become more intermixed, Rose needs grow to exceed those of Tarmaq.

The second part of RQ3 looks at predicting how old the history can grow before it needs to be updated with the latest commits. In order to do so, we first define a set of MAP levels indicating that the history is too old. Specifically, we define the three MAP target levels of 95%, 90% and 80% of the maximum overall MAP value that can be achieved over the different history ages. We perform linear regression with backward elimination, using each of these percentages separately as the response variables, and the demographic information as explanatory variables. For all three response variables and all three techniques, the elimination process removes all the explanatory variables, failing to produce a regression model. This indicates that variations in the demographic variables are not effective predictors of the rate at which a model deteriorates.

In summary, this analysis allows us to answer RQ3: Good values for history length can be predicted. Indeed, a team of developers using Tarmaq can use the regression model found in Table D.10, to predict the amount of system's history that should be used. In contrast, no similar correlation exists for predicting the deterioration of change impact analysis quality based on history age.

5.4 Longitudinal Study

In subsection 5.2 we considered software systems with change histories consisting of up to 35 000 transactions for answering RQ 1.3:

RQ 1.3. *Can we identify an* upper bound *on history length where outdated knowledge starts to negatively affect our analysis causing quality to decrease below acceptable levels?*

For these systems we could not find any evidence that precision degrades when older transactions are used for impact analysis. While 35 000 is a reasonably large change history, software systems which have been built over many years by a plethora of developers see much greater numbers. In this section, we explore the effect of history length for rather extreme values, reaching over 540 000 transactions in the case of the *Linux kernel*.

When considering systems with (very) long histories, it becomes increasingly harder to maintain that they are of approximately equal length. Thus

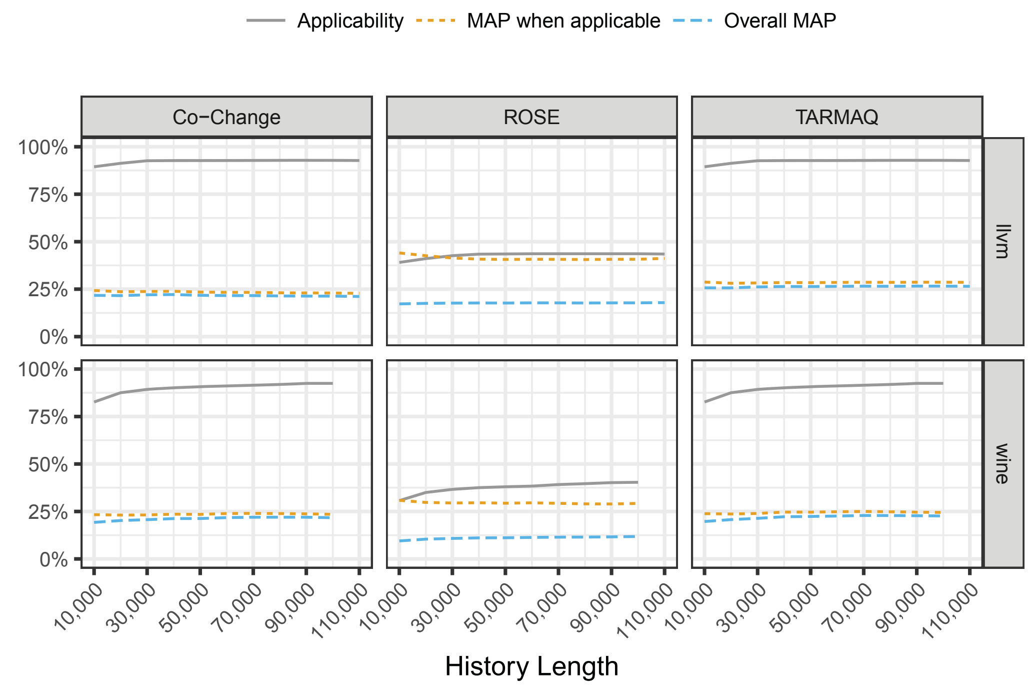


Fig. D.8: Wine and LLVM

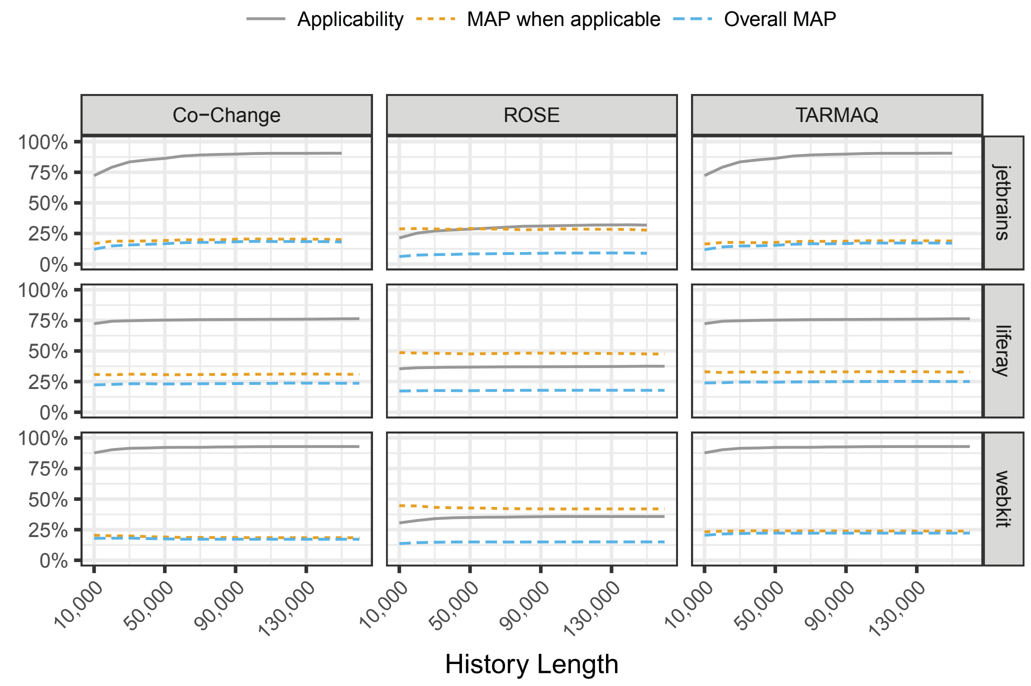


Fig. D.9: Jetbrains, Liferay and Webkit

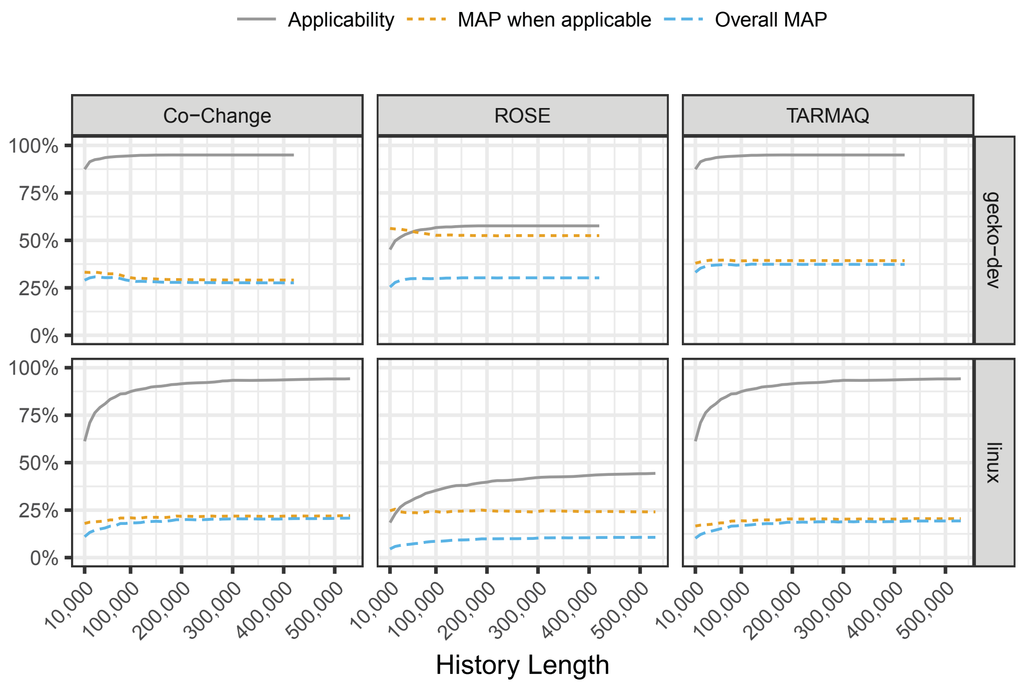


Fig. D.10: Linux and Gecko

for this study, we analyze each software system individually. We first plot the trends in *applicability*, *MAP when applicable* and *overall MAP*, as earlier. The systems are divided into three groups based on approximate history length. In Figure D.8, we plot the trends for *wine* and *llvm* which had around 100 000 transactions, then in Figure D.9 we plot the trends for *jetbrains*, *liferay* and *webkit* which had around 150 000 transactions. Finally, in Figure D.10 we plot the trends for *gecko* and *Linux*, the latter of which reached over 500 000 transactions.

Visually, we see the same trends as in Figure D.5 resurfacing in most cases; a steady rise in applicability up until $30 - 40,000$, and stability in the *MAP when applicable* and *overall MAP* as history length increases further. However, in the case of *jetbrains* and *Linux* it seems to take longer before stability is reached.

To analytically investigate whether long histories deteriorate precision, we first assume that they do. As such we compare the APs of a relatively short but stable history length, against the APs at the maximal available history length with respect to each software system. Throughout our study we have found that history lengths around 30 000 consistently have proved to perform well, we therefore choose this length as our "short length". Our initial hypothesis is thus:

H_0 : The AP at maximal length is not less than the AP at length 30 000.
H_1 : The AP at maximal length is less than the AP at length 30 000.

Subject System	p-value	Cliff's delta	Magnitude of Effect
gecko-dev	0.00	0.14	negligible
jetbrains	0.00	0.48	large
liferay	0.01	0.05	negligible
linux	0.00	0.92	large
llvm	0.10	0.03	negligible
webkit	0.00	0.08	negligible
wine	0.00	0.21	small

Table D.11: Results for running a t-test with alternative hypothesis: "Precision of max history larger than precision at length 30 000"

Perhaps unsurprisingly given our findings so far, we do not obtain significant results for any of the systems, i.e., AP at maximal length is not significantly less than AP at length 30 000. On the contrary, based on the figures it seems more likely that maximal histories actually improves precision compared to the shorter 30 000. We therefore perform the reversed t-test as well:

H_0 : The AP at maximal length is not greater than the AP at length 30 000.
H_1 : The AP at maximal length is greater than the AP at length 30 000.

In this formulation of the t-test we obtain significant results for all software systems but *llvm*. Furthermore, the effect size of the difference as exhibited through *Cliff's delta* supports our earlier visual observation that *Linux* and *jetbrains* continues to see precision gains beyond 30 000 transactions. Cliff's delta also picks up a small effect for *wine*. For the remaining systems the effect is negligible. Furthermore, if we combine the evidence of both one sided t-tests with the negligible effect size, we have sufficient evidence that there is no practical difference between the two history lengths for these systems.

In summary, we have been unable to find any evidence that there is an upperbound to history length above which outdated information negatively affects the precision of impact analysis, even for extremely long history lengths as studied here. Therefore we are confident to answer RQ 1.3 negatively.

5.5 Stability Study

As a software system evolves to the point of maturity one might expect that the quality of change impact analysis on the system also stabilizes. For our final analysis we therefore ask:

RQ 4. *How does choosing a particular* history length *and* history age *affect impact analysis quality throughout the evolution history?*

In order to answer RQ 4 we compare a set of recent queries against a set of older queries and test whether the two distributions can indeed be said to

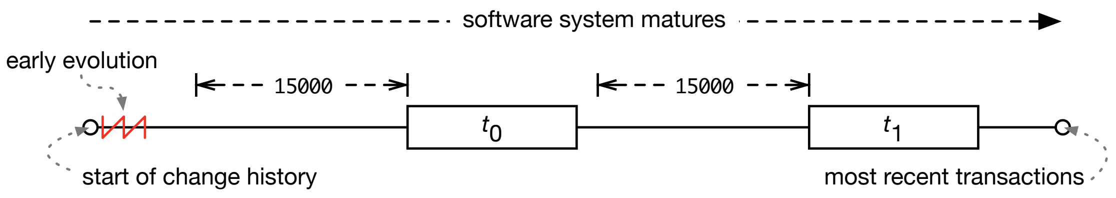


Fig. D.11: The stability study compares two separate periods t_0 and t_1, both outside the early evolution stage, and selected so that queries within each period have access to at least 15 000 previous transactions.

be sufficiently equal. Our design is visualized in Figure D.11 where the two periods in time are labeled t_0 and t_1 respectively. It is important to note that we are not concerned with varying *history length/age* as done for previous research questions, we here solely focus on *time* as an explanatory variable. However, we must still choose a setting for length and age. In terms of *history age*, we set age to zero, as earlier findings in this paper show that aged models significantly degrade the precision. In terms of *history length*, we set this to *15 000* for both t_0 and t_1. A history length of 15 000 strikes a balance between *applicability* (see Figure D.5) and having sufficient space between t_0 and t_1. If a longer *history length* were to be used, t_0 would have to move closer to t_1 in order to guarantee that all queries would have at least 15 000 previous transactions. The further the distance between t_0 and t_1, the stronger we can state that precision has stabilized.

As we need to test for *equivalence* between two distributions, we must first determine a threshold for what we find to be *sufficiently equivalent*. Equivalence tests need a threshold as one cannot expect to have two *exactly equal* distributions in empirical data. Naturally, such a threshold should be anchored in the variation expressed in the data. Therefore, we set the equivalence threshold using the standard deviation. Concretely, we first calculate the mean standard deviation for t_0 and t_1 with respect to each software system, and then set the equivalence threshold equal to the minimum of these two values. This process resulted in a threshold of $\epsilon = 0.034008$, Thus if the difference between values in t_0 and t_1 tend to differ more than ϵ, the likelihood is reduced that we have two equivalent distributions.

Compared to what one could call “normal” hypothesis testing, tests for equivalence reverses the *null hypothesis*, we therefore clearly should state our hypothesis:

H_0 : The average precisions in t_0 and t_1 are different with respect to ϵ
H_1 : The average precisions in t_0 and t_1 are equivalent with respect to ϵ

To test for significant equivalence we use Schuirmann and Westlake’s *two one-sided t-test for equivalence* [28, 29] as implemented in the `R` package `equivalence`. In particular, we apply the function `tost()` to t_0 and t_1 with the equivalence threshold ϵ. Running the test results in a p-value < 0.01, thus we can reject H_0 that performing change impact analysis in the two time periods results in significantly different results. With the difference between t_0 and t_1 of 0.016795, and a 99% confidence interval of $[0.015263, 0.018326]$, we are well below the threshold of 0.034008. To be clear, there is a 1% chance that our confidence interval does not contain the true difference given the population sample.

Given the results we can conclude that as software systems reach a certain maturity, here defined as having a change history consisting of at least 15 000 transactions, the quality of change impact analysis should remain consistent in terms of precision. In essence this tells us that most implementation tasks goes in *waves* where related artifacts tend to be changed within the same time window. However, for this study we used a relatively large time window of 15 000 which perhaps could be both reduced in early evolution, and increased in later evolution. We plan to replicate our study with a greater range of time windows in the future.

6 Threats to Validity

Realism of Scenarios used in Evaluation: We evaluate mining-based change impact analysis by establishing a ground truth from historical transactions, randomly splitting them into a query and an expected outcome of a certain size. However, this approach does not account for the actual order in which changes were made before they were committed together to the versioning system. As a result, it is possible that our queries contain elements that were actually changed later in time than elements of the expected outcome. This cannot be avoided when using co-change data obtained from a versioning system, since the timing of individual changes is lost in that data. It *can* be addressed by using another source of co-change data, such as developer interaction with an IDE, but the invasiveness of that type of data collection makes that there are only very limited data-sets available, which would prevent a study as comprehensive as presented here. Moreover, since the evolutionary couplings that are at the basis of our change impact analysis form a bi-directional relation, the actual order in which changes were made before they were committed has no impact on the result, as goal is not to re-enact the actual timeline of changes, but to establish a ground truth w.r.t. related artifacts.
Variation in software systems: We evaluated the impact of history length and age on mined change impact by studying two industrial systems and 17 large open source systems. These systems vary considerably in size and frequency of transactions (commits), which should provide an accurate picture of the performance of hyper-rules in various contexts. However, despite our

careful choice, we are likely not to have captured all variations.
Commits as basis for evolutionary coupling: The evaluation in this paper is grounded in predictions based on the analysis of patterns found in the change histories. The transactions that make up the change histories are however not in any way guaranteed to be “correct” or “complete”, in the sense that they represent a coherent unit of work. Non-related artifacts may be present in the transactions, and related artifacts may be missing from the transactions. However, the included software-systems in our evaluation all (except KM) use Git for version control. As Git provides developers with tools for history-rewriting, we do believe that this might cause more coherent transactions.
Equal weight for all commits: In our experiment, all transactions from change history are given equal weight while mining change impact. A compelling alternative viewpoint is that more recent transactions are more relevant for current developments and should therefore be given higher weight than older transactions. Similarly, one could argue that, because of their knowledge about the system, transactions committed by code developers should be given higher weight than transactions committed by occasional contributors. For the study described in this paper, we do not include such orthogonal weighing scenarios in our experiments because of their interaction with several of our research questions, such as the length at which considering a longer history would no longer benefit impact analysis quality, or the length at which considering a longer history would start decreasing impact analysis quality due to the inclusion of outdated information. Moreover, the systems considered in this study use a contribution process that includes code reviewing based on pull requests before taking in changes from occasional contributors. We believe this process largely removes the differences between transactions committed by core developers and transactions that originate from occasional contributors but were accepted after review.
Implementation: We implemented and thoroughly tested all algorithms, aggregators and interestingness measures studied in this paper in Ruby. However, we can not guarantee the absence of implementation errors which may have affected our evaluation.

7 Related Work

Software repository mining literature [11, 21, 22] frequently alludes to the notion that learning from a too short, or an overly long history harms the outcome, either because not enough knowledge can be uncovered, or because outdated information introduces noise. However, except for some smaller experiments by Zimmermann [11], the impact of these effects has not been systematically investigated.

Similarly, authors in the field of association rule mining have stated the need to investigate sensitivity to algorithm parameters (e.g., transaction filter size used, choice of interestingness measure) [30–33], but we have not found work that discusses sensitivity to the number of transactions used for mining (i.e., our history length), or to aging of transactions.

Parameters in Mining Change Impact: In the context of software change impact analysis, several studies remark on the importance of discarding from the history large change sets which are likely to contain unrelated artifacts. For example, Kagdi et al. [24], Zimmermann et al. [11] and Ying et al. [12] propose to filter out transactions larger than 10, 30, and 100 items, respectively. However, none of this work reports how the threshold was chosen nor does it discuss the impact of different values on recommendation quality. In previous work [25], we systematically explored the effect of filtering size on the quality of change impact analysis, and found that filtering transactions larger than eight items yields the best result for similar systems as considered in this paper.

Characteristics of the Change History: Over the years, several studies proposed strategies to group transactions in the revision history of software projects [11, 13, 34]. The reason for doing so is that a developer might accidentally commit an incomplete transaction, and modify the remaining files related to the same change in a subsequent transaction. As a consequence, a single change set might be scattered across several transactions in the change history. Nevertheless, in modern version control systems, transactions are stashed in the user local repository and finalized at a later stage, reducing the risk of committing incomplete transactions.

In contrast, whether properties such as average commit size and frequency affect the quality of software recommendations is a relatively less studied subject. In this direction, German carried out an empirical study on several open source projects, finding that the revision history of most systems contains mostly small commits [35]. Alali et al. also investigated the total number of lines modified in the files, and the total number of hunks with line changes [36]. Kolassa et al. performed a similar study on commit frequency, reporting an average inter-commit time around three days [37]. However, none of these studies investigates how characteristics of the change history affect the quality of change recommendations.

Characteristics of the Change Set: Targeted association rule mining approaches drive the generation of rules by a *query* supplied by the user [20]. In general, characteristics of the query can effect the precision of recommendations. For example, Rolfsnes et al. found a particular class of queries, strongly related to query size, for which the most common targeted association rule mining approaches cannot generate recommendations [10]. In other work, Hassan and Holt investigated the effectiveness of evolutionary coupling in predicting change propagation effects resulting from source code changes, but did not evaluate whether the size of transactions in the history affects the

quality of the predictions generated [7].
Aged histories in evaluation: For the purpose of evaluating change impact analysis or change recommendation techniques, it is common practice to split the change history into training and test sets. The training set can either be treated as a static prediction model [12], or be continuously updated with respect to the chosen transaction from the test set [11]. If treated as a static model this means that the model will be aged differently with respect to each transactions in the test set, and as we have seen in RQ2 aging affects impact analysis quality. Therefore, any study involving history splitting should take aging into consideration. Since we have seen that aging can only lead to deterioration of impact analysis quality, we suggest evaluation setups where the prediction model is updated for every transaction in the test set (i.e., an age of zero).

8 Concluding Remarks

This paper presents a systematic study of the effects of history length and age on 19 different software systems. Key findings include that as history length increases, the MAP value also increases, although this increase diminishes at around 15 000 commits. Moreover, the applicability also increases with increased history length, but seems to top out around 25 000 commits. We found that the impact of age on the MAP value is very significant, as even very little aging yields a strong, basically exponential decrease.

Even in our longest studies of up to 540 000 commits, we found no evidence for the commonly held belief that there is an upper-bound to the history that can be used for mining evolutionary coupling before outdated knowledge starts to negatively affect the recommendation quality.

In addition to these studies, to provide a better understanding of the impact of history length and age on the quality of change impact analysis, we also derive a prediction model for the length of the history that should be used with a given system. This prediction model is a function of system demographics, specifically the average number of artifacts in a commit, the median inter-commit time, and the number of unique artifacts in the history. We found no corresponding prediction model for system age, which likely reinforces the rapid detrimental effects of aging.

Finally, we found that the mining algorithms are *stable* with respect to the effects of choosing a particular history length and history age on the impact analysis quality throughout the evolution history. This means that for a system that has matured beyond its chaotic youth, an optimal history length can be computed *once* (using our prediction model) and subsequently need not to be updated as the system matures further.

8.1 Future Work

Looking forward, an interesting avenue of investigation would be to assess the impact of rule aggregation [26] on change impact analysis quality. This analysis was not considered for the current paper to maintain conceptual integrity, and avoid the inclusion of too many orthogonal topics.

Moreover, based on our findings related to the impact of very recent transactions and history age on change impact analysis quality, it would be interesting to experiment with (a) an alternative targeted association rule mining algorithm that does not consider a fixed history length but instead uses an adaptive approach, growing the history until (a number of) applicable transactions are found, and (b) alternative strategies for association rules generation that take age into account, for instance by assigning higher weight to more recent transactions.

Finally, it would be interesting to analyze how aspects of the development process affect the evolutionary coupling and change impacts that are mined from historical co-change data. For example, should transactions from all contributors to a project be considered equally, or would it be beneficial to give higher weight to transactions from core team members. Another aspect to investigate is the relation between the velocity of a development project, or even the typical time between commits, and good values for history size and age.

Acknowledgement: This work is supported by the Research Council of Norway through the EvolveIT project [3] and the Certus SFI [4]. Dr. Binkley was supported by NSF grant IIA-1360707 and a J. William Fulbright award.

References

[1] S. Bohner and R. Arnold, *Software Change Impact Analysis*. CA, USA: IEEE, 1996.

[2] J. Law and G. Rothermel, "Whole Program Path-Based Dynamic Impact Analysis," in *International Conference on Software Engineering (ICSE)*. IEEE, 2003, pp. 308–318. [Online]. Available: http://dl.acm.org/citation.cfm?id=776816.776854

[3] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, "Chianti: a tool for change impact analysis of java programs," in *ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2004, pp. 432–448. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1035292.1029012

[3] #221751/F20

[4] #203461/030

[4] M.-A. Jashki, R. Zafarani, and E. Bagheri, "Towards a more efficient static software change impact analysis method," in *ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE)*. ACM, 2008, pp. 84–90. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1512475.1512493

[5] A. R. Yazdanshenas and L. Moonen, "Crossing the boundaries while analyzing heterogeneous component-based software systems," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011, pp. 193–202. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2011.6080786http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6080786

[6] A. Podgurski and L. Clarke, "A formal model of program dependences and its implications for software testing, debugging, and maintenance," *IEEE Transactions on Software Engineering*, vol. 16, no. 9, pp. 965–979, 1990. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=58784

[7] A. E. Hassan and R. Holt, "Predicting change propagation in software systems," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2004, pp. 284–293. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1357812

[8] G. Canfora and L. Cerulo, "Impact Analysis by Mining Software and Change Request Repositories," in *International Software Metrics Symposium (METRICS)*. IEEE, 2005, pp. 29–37. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1509307http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1509307

[9] M. B. Zanjani, G. Swartzendruber, and H. Kagdi, "Impact analysis of change requests on source code based on interaction and commit histories," in *International Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 162–171. [Online]. Available: http://dl.acm.org/citation.cfm?id=2597096http://dx.doi.org/10.1145/2597073.2597096

[10] T. Rolfsnes, S. Di Alesio, R. Behjati, L. Moonen, and D. W. Binkley, "Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis," in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, mar 2016, pp. 201–212. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7476643

[11] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software*

Engineering, vol. 31, no. 6, pp. 429–445, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1463228

[12] A. T. T. Ying, G. Murphy, R. T. Ng, and M. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1324645

[13] H. Kagdi, S. Yusuf, and J. I. Maletic, "Mining sequences of changed-files from version histories," in *International Workshop on Mining Software Repositories (MSR)*. ACM, 2006, pp. 47–53. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1137983.1137996

[14] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD International Conference on Management of Data*. ACM, 1993, pp. 207–216. [Online]. Available: http://portal.acm.org/citation.cfm?doid=170035.170072

[15] L. Moonen, S. Di Alesio, T. Rolfsnes, and D. W. Binkley, "Exploring the Effects of History Length and Age on Mining Software Change Impact," in *IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, sep 2016, pp. 207–216.

[16] S. Eick, T. L. Graves, A. Karr, J. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 1–12, 2001. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=895984

[17] M. Gethers, H. Kagdi, B. Dit, and D. Poshyvanyk, "An adaptive approach to impact analysis from change requests to source code," in *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, nov 2011, pp. 540–543. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6100120

[18] R. Robbes, D. Pollet, and M. Lanza, "Logical Coupling Based on Fine-Grained Change Information," in *Working Conference on Reverse Engineering (WCRE)*. IEEE, 2008, pp. 42–46. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4656392

[19] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 1998, pp. 190–198. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=738508

[20] R. Srikant, Q. Vu, and R. Agrawal, “Mining Association Rules with Item Constraints,” in *International Conference on Knowledge Discovery and Data Mining (KDD)*. AASI, 1997, pp. 67–73.

[21] T. L. Graves, A. Karr, J. Marron, and H. P. Siy, “Predicting fault incidence using software change history,” *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653–661, jul 2000. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=859533

[22] A. E. Hassan, “The road ahead for Mining Software Repositories,” in *Frontiers of Software Maintenance*. IEEE, 2008, pp. 48–57. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4659248

[23] A. Alali, “An Empirical Characterization of Commits in Software Repositories,” Ms.c, Kent State University, 2008.

[24] H. Kagdi, M. Gethers, and D. Poshyvanyk, “Integrating conceptual and logical couplings for change impact analysis in software,” *Empirical Software Engineering*, vol. 18, no. 5, pp. 933–969, oct 2013. [Online]. Available: http://link.springer.com/10.1007/s10664-012-9233-9

[25] L. Moonen, S. Di Alesio, D. W. Binkley, and T. Rolfsnes, “Practical Guidelines for Change Recommendation using Association Rule Mining,” in *International Conference on Automated Software Engineering (ASE)*. Singapore: IEEE, sep 2016.

[26] T. Rolfsnes, L. Moonen, S. Di Alesio, R. Behjati, and D. W. Binkley, “Improving change recommendation using aggregated association rules,” in *International Conference on Mining Software Repositories (MSR)*. ACM, 2016, pp. 73–84. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2901739.2901756

[27] R. Baeza-Yates and B. Ribeiro-Neto, *Modern information retrieval*. ACM, 1999.

[28] D. Schuirmann, “On hypothesis testing to determine if the mean of a normal distribution is contained in a known interval.” *Biometrics*, 1981.

[29] W. Westlake, “Response to T.B.L. Kirkwood: bioequivalence testing - a need to rethink,” *Biometrics*, vol. 37, pp. 589–594, 1981.

[30] Z. Zheng, R. Kohavi, and L. Mason, “Real world performance of association rule algorithms,” in *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2001, pp. 401–406. [Online]. Available: http://portal.acm.org/citation.cfm?doid=502512.502572

[31] W. Lin, S. A. Alvarez, and C. Ruiz, "Efficient Adaptive-Support Association Rule Mining for Recommender Systems," *Data Mining and Knowledge Discovery*, vol. 6, no. 1, pp. 83–105, 2002. [Online]. Available: http://link.springer.com/10.1023/A:1013284820704

[32] N. Jiang and L. Gruenwald, "Research issues in data stream association rule mining," *ACM SIGMOD Record*, vol. 35, no. 1, pp. 14–19, mar 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1121995.1121998

[33] O. Maimon and L. Rokach, *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Springer, 2010. [Online]. Available: http://books.google.com/books?hl=en{&}lr={&}id=S-XvEQWABeUC{&}oi=fnd{&}pg=PR21{&}dq=Data+Mining+and+knowledge+discovery+handbook{&}ots=LBVkfoBx6S{&}sig=u6c1n2kopRhLrbpg5M0FhvYhFqk{%}5Cnhttp://www.springerlink.com/index/10.1007/978-0-387-09823-4http://link.springer.com/1

[34] F. Jaafar, Y.-G. Guéhéneuc, S. Hamel, and G. Antoniol, "Detecting asynchrony and dephase change patterns by mining software repositories," *Journal of Software: Evolution and Process*, vol. 26, no. 1, pp. 77–106, jan 2014. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/smr.504/fullhttp://doi.wiley.com/10.1002/smr.1635

[35] D. M. German, "An empirical study of fine-grained software modifications," *Empirical Software Engineering*, vol. 11, no. 3, pp. 369–393, 2006.

[36] A. Alali, H. Kagdi, and J. I. Maletic, "What's a Typical Commit? A Characterization of Open Source Software Repositories," in *International Conference on Program Comprehension (ICPC)*. IEEE, 2008, pp. 182–191. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4556130

[37] C. Kolassa, D. Riehle, and M. A. Salim, "The empirical commit frequency distribution of open source projects," in *International Symposium on Open Collaboration (WikiSym)*. ACM, 2013, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2491055.2491073

Paper E

Predicting Relevance of Change Recommendations

Thomas Rolfsnes, Leon Moonen and Dave W. Binkley

Accepted for publication in the main research track of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), October 30 - November 3, 2017.

© 2017 IEEE
The layout has been revised.

Abstract

Software change recommendation seeks to suggest artifacts (e.g., files or methods) that are related to changes made by a developer, and thus identifies possible omissions or next steps. While one obvious challenge for recommender systems is to produce accurate *recommendations, a complimentary challenge is assessing the* relevance *of a recommendation. We address this complimentary challenge for recommendation systems based on* evolutionary coupling. *Such systems use* targeted association-rule mining *to identify* relevant patterns *in a software system's change history. Traditionally, this process involves ranking artifacts using* interestingness measures *such as* confidence *and* support. *However, these measures often fall short when used to assess recommendation relevance.*

We propose the use of random forest classification models to assess recommendation relevance. This approach improves on past use of various interestingness measures by learning from previous change recommendations. We empirically evaluate our approach on fourteen open source systems and two systems from our industry partners. Furthermore, we consider two complementing mining algorithms: Co-Change *and* Tarmaq*. The results find that random forest classification significantly outperforms previous approaches, receives lower Brier scores, and has superior trade-off between precision and recall. The results are consistent across software system and mining algorithm.*

1 Introduction

When software systems evolve, the interactions between source code entities grow in number and complexity. As a result, it becomes increasingly challenging for developers to foresee and reason about the overall effect of a change to the system. One proposed solution, change impact analysis [1], aims to identify software artifacts (e.g., files, methods, classes) affected by a given set of changes. The impacted artifacts form the basis for *change recommendation*, which suggests to an engineer artifacts potentially missed while implementing a change to the code.

One promising approach to change recommendation aims to identify potentially relevant items based on *evolutionary* (or *logical*) *coupling*. This approach can be based on a range of granularities of co-change information [2–4] as well as code-churn [5] or even interactions with an IDE [6]. Change recommendation based on evolutionary coupling has the intriguing property that it effectively taps into the inherent knowledge that software developers have regarding dependencies between the artifacts of a system. Our present work considers co-change information extracted from a version control system such as Git, SVN, or Mercurial.

One challenge faced by all recommender systems are false positives. This challenge becomes acute if developers come to believe that automated tools are "mostly wrong" [7]. Cleary algorithms with higher accuracy will help address this challenge. However, we can also address this challenge with algorithms that assess the *relevance* of a proposed recommendation. In fact the two approaches are complimentary. In this paper we hypothesize that *history aware* relevance prediction that exploits earlier change recommendations to assess the relevance of a current recommendation can help mitigate the challenge of false positives.

Our approach consists of training a *random forest classification model* [8] on previous change recommendations with *known relevance*. The model can then be used to give a *single likelihood estimate* of the relevance of future change recommendations. Automatic assessment of recommendation relevance, frees developers from having to perform the potentially time-consuming task. Our work therefore facilitates tooling which can automate the change recommendation process, only notifying the developer when relevant recommendations are available. Furthermore, our approach compliments existing research work on improving mining algorithm accuracy as it can be used regardless of the mining algorithm used to generate a recommendation.

Contributions: **(a)** We present twelve features describing aspects of *change sets*, *change histories*, and generated *change recommendations*. These features are used to build a random forest classification model for *recommendation relevance*. **(b)** We assess our model in a large empirical study encompassing sixteen software systems, including two from our industry partners *Cisco* and *Kongsberg Maritime*. Furthermore, change recommendations are generated using both CO-CHANGE and TARMAQ to assess the external validity of our approach. **(c)** We evaluate the importance of each of the twelve features used in our relevance classification model.

2 Overall Approach

The overarching goal of this paper is to attach appropriate *relevance scores* to change recommendations based on association rule mining. We consider this question of relevance from a developer viewpoint, where "relevant" means "useful for the developer". We therefore consider a change recommendation relevant if it contains a correct artifact as one of its top ten highest ranked artifacts.

We propose an approach that uses *classification based on random forests* [8] to learn from previous change recommendations in order to assess the current one. Thus, the relevance of a current change recommendation is evaluated based on the *known relevance* of historically similar and dissimilar recommendations.

Traditionally, the same interestingness measure used to rank the artifacts of a recommendation are also used to assess its relevance [9–11]. For example, an interestingness measure might weight artifacts on a range from 0 to 1, enabling an internal ranking. Given the ranking it is then up to the user to assess whether the recommendation is relevant. Naturally, if the top ranked artifacts have received weights close to the maximum (1 in this example), the recommendation is assumed relevant and will likely be acted upon. Recent work found that, in the context of software change recommendation, Agrawal's *confidence* interestingness measure [12] performs among the top-in-class when compared to more complex measures [13]. Considering this result, we use confidence as a baseline and set out to compare the relevance predictions given by our proposed approach against those based on confidence:

RQ 1. *Does classification based on random forests improve upon the use of confidence as a relevance classifier?*

To train our classification model, a range of features must be introduced that describe attributes related to a change recommendation. The better these features capture *key attributes* the better we can learn from previous recommendations and consequently the better we can assess the relevance of a current recommendation. Thus our second research question is

RQ 2. *What are the most important features for classifying change recommendation relevance?*

In our study we use random forests for their proven performance [14] and intrinsic ability to assess variable (feature) importance [8]. By answering our two research questions we seek to uncover whether change recommendation relevance is a viable venue for further research. If so, our approach may prove to be an important compliment to existing algorithms for change recommendation. As both improve, we gain both better recommendations alongside higher confidence in those recommendations.

3 Related Work

Recommendation Relevance: A shared goal in recommendation systems is uncovering *interesting* findings in a dataset, which means that an important research question concerns what actually characterizes the *interestingness* of a finding. Over the years, numerous measures for interestingness have been proposed [9–11, 15, 16]. A recent evaluation of 39 of these measures in the context of software change recommendation found that the traditional measures of *confidence* and *support* perform just as well as more recent and often more complex measures [13].

Cheetham and Price evaluated *indicators* (features) that can be used to provide confidence scores for *case based reasoning systems* [17, 18]. To provide a recommendation for a new case, the *k-nearest neighbor* algorithm was used, thus the evaluated features were tightly woven with the kind of output that the *k*-nearest neighbor algorithm produces. Example features include the "Number of cases retrieved with best solution" and "Similarity of most similar case." The features were tested for importance using leave-one-out testing in combination with the decision tree algorithm C4.5. In comparison, our use of random forests avoids the need for leave out testing of features as feature importance is internally accounted for.

Le et al. proposed an approach for predicting whether the output of a bug localization tool is relevant [19, 20]. As for this paper, an output is considered relevant if a true positive is part of the top 10 recommended artifacts. While *change recommendation* can be seen as stopping faults before they happen, *bug localization* is a complementary approach for already existing faults. State of the art bug localization is based on comparing normal and faulty execution traces (spectrum based). In order to predict relevance, Le et al. identified 50 features related primarily to traces, but also considered the weights of recommended (suspicious) artifacts. These features were then used to train a classification model for relevance based on SVM.

Rule aggregation, clustering, and filtering: Rolfsnes et al. propose aggregation of association rules to combine their evidence (or interestingness) [21]. Aggregation likely increases recommendation relevance: for example, consider three rules, one recommending A with confidence 0.8 and two recommending B with confidence 0.7 and 0.6 respectively. Traditional approaches would use the highest ranking rule and thus prioritize A over B. Rule aggregation enables combining the evidence for B and thus leads to recommending B over A.

Several authors propose methods to discover the most informative rules in a large collection of mined association rules, by either clustering rules that convey redundant information [22–24], or by pruning *non-interesting* rules [25, 26]. The overall idea is that the removal of rules will reduce the noise in the recommendations made using the remaining rules. However, in contrast to the work by Rolfsnes et al. and the approach proposed in this paper, recommendations will be based on only part of the available evidence. It remains to be seen if this affects relevance.

Parameter tuning: Recent research highlighted that the configuration parameters of data mining algorithms have a significant impact on the quality of their results [27]. In the context of association rule mining, several authors have highlighted the need for thoughtfully studying the impact of parameter settings on the quality of the generated rules [28–30].

Moonen et al. investigated how the quality of software change recommendation varied depending on association rule mining parameters such as transaction filtering threshold, history length, and history age [13, 31]. In contrast to that work, which focused on configurable parameters of the algorithm, this paper considers *non-configurable features* of the query, the change history, and the recommendation history.

4 Generating Change Recommendations

4.1 Software Change Recommendation

Recommender (or recommendation) engines are information filtering systems whose goal is to predict *relevant* items for a specific purpose [32]. A common use of recommender systems is in marketing where these systems typically leverage a shopper's previous purchases and the purchases of other shoppers to predict items of potential interest to the shopper.

In the context of software engineering, these systems typically leverage a developer's previous changes together with the changes made by other developers to predict items of interest. *Software change recommendation* takes as input a set of changed entities, referred to as a *change set* or *query*, and predicts a set of entities that are also likely in need of change. These entities may be any software *artifact*, such as files, methods, models, or text documents. The study described in this paper considers both files and methods as potential artifacts. We extract these artifacts from the version control history of a software system. Thus, software recommendation helps answering questions such as *"Given that files f_1 and f_2 and method m changed, what other files and methods are likely to need to be changed?"*

A common strategy for change recommendation is to capture the *evolutionary coupling* between entities [2]. In this application, entities are considered coupled iff they have changed together in the past. The key assumption behind evolutionary coupling is that the more frequently two or more entities change together, the more likely it is that when one changes, the others will also have to be changed. In the context of software change recommendation, evolutionary coupling is commonly captured using association rule mining [33].

4.2 Targeted Association Rule Mining

Association rule mining is an unsupervised learning technique aimed at finding patterns among items in a data set [12]. Association rule mining was first applied for market basket analysis, to find patterns (*rules*) describing items people typically purchase together. In this context, the data set is expressed as a list of transactions, where each transaction consists of a set of items. For example, in the domain of grocery stores, items likely include products such as "milk" and "cookies", and mining a rule such as "cookies" $\rightarrow$ "milk", uncovers the tendency of people who buy cookies (the rule *antecedent*) to also buy milk (the rule *consequent*). This provides valuable knowledge, for example suggesting that placing these grocery items in close proximity will increase sales.

Shortly after the introduction of association rule mining, Srikant et al. acknowledged that for most applications only a few specific rules are of practical value [34]. This led to the development of *constraint-based rule mining* where only rules that satisfy a given constraint are mined. Typically, constraints specify that particular items must be involved in the rule's antecedent. For example, consider a software engineer who recently made a change to file x. A constraint could specify that rule antecedents must contain x, thus limiting recommendation to those involving x. Constraints are usually specified by the user in the form of a *query*, at which the mining process is said to be *targeted*. The resulting *Targeted Association Rule Mining Algorithms* filter from the history all transactions unrelated to the query, producing a more focused set of rules. Doing so provides a significant reduction in execution time [34].

To rank the resulting rules, numerous *interestingness measures* have been proposed [11, 15]. Such measure attempt to quantify the likelihood that a rule will prove useful. The first interestingness measures introduced, *frequency*, *support*, and *confidence*, are also the most commonly used [12]. It is worth formalizing these three. Each is defined in terms of a *history*, $\mathcal{H}$, of transactions and an association rule $A \to B$, where A and B are disjoint sets of items. To begin with rule *frequency* is the number of times the antecedent and consequent have changed together in the history:

Definition 1 (Frequency).

$$\mathit{frequency}(A \to B) \overset{\mathit{def}}{=} |\{T \in \mathcal{H} : A \cup B \subseteq T\}|$$

Second, the *support* of a rule is its relative frequency with respect to the total number of transactions in the history:

Definition 2 (Support).

$$\mathit{support}(A \to B) \overset{\mathit{def}}{=} \frac{\mathit{frequency}(A \to B)}{|\mathcal{H}|}$$

Finally, *confidence* is its relative frequency of the rule with respect to the number of historical transactions containing the antecedent A:

Definition 3 (Confidence).

$$\mathit{confidence}(A \to B) \overset{\mathit{def}}{=} \frac{\mathit{frequency}(A \to B)}{|\{T \in \mathcal{H} : A \subseteq T\}|}$$

Support and confidence are often combined in the *support-confidence framework* [12], which first discards rules below a given support threshold and then ranks the remaining rules based on confidence. Thresholds were originally required to minimize the number of potential rules, which can quickly grow unwieldy. However, the constraints introduced by targeted association rule mining greatly reduce the number of rules and thus do not depend on a support threshold for practical feasibility.

4.3 Association Rule Mining Algorithms

A targeted association rule mining algorithm takes as input a query Q and restricts the antecedents of the generated rules to be various subsets of Q. The variation here comes from each algorithm attempting to best capture relevant rules. Consider, for example, the query $Q = \{a, b, c, d\}$. Potential rules include $a \rightarrow X$ and $b, c \rightarrow Y$. In fact, the set of possible antecedents is given by the powerset of Q.

One of most well known algorithms, Rose, limits the number of rules by requiring that the antecedents are equal to the query, Q [33]. Thus for $\{a, b, c, d\}$, Rose rules are all of the form $a, b, c, d \rightarrow X$, where X is only recommended if there exists one or more transactions where X changed together with *all* the items of the query. At the other end of the spectrum, Co-Change partitions Q into singletons and considers only rules for each respective singleton [35]. Thus it produces rules of the form $a \rightarrow x$ and $b \rightarrow x$.

While Rose and Co-Change makes use of the largest and smallest possible rule antecedents, Tarmaq identifies the largest subset of Q that has changed with something else in the past [36]. Thus Tarmaq may return the same rules as Co-Change (when the history is made up of only two-item transactions) or the same rules as Rose (when Q is a proper subset of at least one transactions). However, Tarmaq is also able to exploit partial evidence (e.g., when the history includes transactions larger than two, but none that Rose can make use of).

While it may not be immediately evident, Tarmaq is defined such that its recommendations are identical to those of Rose, *when* Rose is able to produce a recommendation. On the other hand, Tarmaq can produce recommendations far more often than Rose [36]. As a result, we performed our empirical study using Co-Change and Tarmaq, as the behaviour of Rose is subsumed by that of Tarmaq. As Co-Change mines only singleton rules and Tarmaq potentially mines rules which maximize the antecedent with respect to the query, they together cover a large range of possible recommendations.

5 Overview of Model Features

This section introduces the features that we use to build our random forest classification model. We consider three categories of features:

- features of the query,
- features of the change history
- features of the recommendation.

It is worth noting that the features describing the query are known a priori, while features of the change history and the change recommendation are only known after a recommendation has been generated. Fortunately, a change recommendations can be generated in mere milliseconds and the corresponding feature set can therefore be included without incurring undo computational expense.

5.1 Features of the Query

Query Size: The first feature of a query we consider is simply its size. For example, if a single method is changed the query size is 1, if two different methods are changed, the query size is 2 and so on. Furthermore, some files may not be able to be parsed for fine-grained change information, changes to these files only ever increase the query size by 1 in total. Throughout the rest of the paper we use the term *artifact* as a generic way of referring to both (unparsable) files and (parsable) methods. We hypothesize that *query size* may be important for relevance as when it increases one of the following is likely occurring: **(a)** The developer is working on a complex feature, requiring code updates in several locations. Here increased query size indicates *specificity*. **(b)** On the other hand, if a developer is not compartmentalizing the work and thus is working on multiple features at the same time, increased query size indicates *chaos* as unrelated artifacts are added to the same commit.
Number of artifacts added: If a new file or new method is added, we know that nothing has changed together with it previously, thus from an *evolutionary perspective*, the new artifact is uncoupled from all other artifacts. The presence of new artifacts in combination with known artifacts adds uncertainty and is therefore considered as a feature.
Number of methods/files changed: We record the granularity of changes in two separate features: the number of methods changed and the number of files changed. For example, if the methods $m1$ and $m2$ change in the file $f1$, and the method $m3$ change in the file $f2$, we record that 3 methods and 2 files have changed. By including these metrics of query granularity we hope to capture the *specificity* of the corresponding change recommendation.

5.2 Features of the Change History

Whenever an existing artifact, *a*, is changed, a list of *relevant transactions* can be extracted from the change history. A transaction is relevant if it contains the changed artifact. From these transactions mining algorithms identify other artifacts that typically changed with *a*, forming the basis for the resulting change recommendation.

Number of relevant transactions: The number of relevant transactions is the number of transactions with at least one artifact from the query. This metrics provides a measure of the *churn rate* (i.e., how often the artifacts change).

Mean size of relevant transactions: While the number of relevant transactions tells us how often the artifacts found in a query change, it does not tell us how often they change with other artifacts, the *mean size of relevant transactions* attempts to capture this feature.

Mean/median age of relevant transactions: Two age related features included involve the change in dependencies between artifacts as software evolves (e.g., because of a refactoring) and *code decay*, where artifacts become "harder to change" (a known phenomena in long lived software systems [37]). The feature "age of relevant transactions" attempts to capture these two age related aspects. Note that two features are actually used: the *mean age* and the *median age*.

Overlap of query and relevant transactions: If there are transactions that exhibit large overlap with the query, this might indicate highly relevant transactions [33]. We capture this through the "overlap percentage". Note that the percentage reports the single largest match rather than the mean.

5.3 Features of the Recommendation

A recommendation boils down to a prioritized list of association rules giving the *potentially affected* artifacts. However, different mining algorithms may return different lists. While the features described so far are independent of the mining algorithm, in this section we consider features that are aspects of the recommendation and thus the particular algorithm used.

The Confidence and Support of Association Rules: A recommendation is constructed from *association rules* of the form A $\rightarrow$ B. Here "*A*" includes changed artifacts while "*B*" the recommended artifacts. To be able to distinguish rules, weights can be provided through *interestingness measures*. One way of providing these weights uses the support and confidence interestingness measures (Definitions 2 and 3). Traditionally, interestingness measures are used in isolation to judge whether a recommendation is relevant [33, 36, 38]. In this paper we extend their use by considering aggregates of the top 10 rules in order to indicate recommendation relevance. We include three aggregates: The *top 10 mean confidence*, the *top 10 mean support* and the *maximum* *confidence*. Each feature is meant to capture the likelihood of whether there exist at least one relevant artifact in the top ten.

Number of Association Rules: If a recommendation consists of a large number of rules, two non-mutually exclusive situations may exist: **(a)** the query is large and the contained artifacts have changed with something else in the past, or **(b)** at least one artifact of the query has changed with a large number of other artifacts in the past. In either case, a large recommendation is a symptom of non-specificity and may thus prove a valuable feature for classifying true negatives.

6 Experiment Design

Our empirical study is designed to answer one primary question: *can we predict if a recommendation contains relevant artifacts?* To answer this question we generate a large *recommendation oracle*, over which we train random forest classification models using the features described in section 5. Finally, we evaluate the resulting models by comparing their performance against two confidence based predictions of relevance. These aim to function as a baseline for whether a developer would act upon a given recommendation.

1. **Maximum Confidence:** a recommendation is predicted as relevant if the confidence of the artifact with the highest confidence is larger than a given threshold.

2. **Mean Confidence 10:** a recommendation is predicted as relevant if the mean confidence of the top ten artifacts is greater than a given threshold.

The rationale behind *maximum confidence* mimics a developer who is only willing to consider a recommendation's highest ranked artifact, while that of *mean confidence 10* mimics a developer who is willing to consider the recommendation's top ten artifacts.

Our study encompasses change recommendations generated from the change history of sixteen different software systems with varying characteristics. Two of these systems come from our industry partners, *Cisco* and *Kongsberg Maritime*. *Cisco* is a worldwide leader in the production of networking equipment, We analyze the software product line for professional video conferencing systems developed by *Cisco*. *Kongsberg Maritime* is a leader in the production of systems for positioning, surveying, navigation, and automation of merchant vessels and offshore installations. We analyze the common software platform *Kongsberg Maritime* uses across various systems in the maritime and energy domain.

The other fourteen systems are the well known open-source projects reported in Table E.1 along with change history statistics illustrating their diversity. The table shows that the systems vary from medium to large size, with over 280 000 unique files in the largest system. For each system, we extracted up to the 50 000 most recent transactions. This number of transactions covers vastly different time spans across the systems, ranging from almost twenty years in the case of HTTPD, to a little over ten months in the case of the Linux kernel. In the table, we report the number of unique files changed throughout the 50 000 most recent transactions, as well as the the number of unique artifacts changed. These artifacts include, in addition to file-level changes, method-level changes for certain languages.[1] Finally, the last column of Table E.1 shows the programming languages used in each system, as an indication of heterogeneity.

The remainder of this section first explains the setup used to generate change recommendations using Co-Change and Tarmaq. It then details how these recommendations are used to train and evaluate models for relevance prediction. The results of our study are presented in section 7.

6.1 Generating Change Recommendations

Creating gold standards

The change history of a software system exactly describes, transaction by transaction, how to (re)construct the current state of the system. Consequently, we can assume the majority of the time each transaction has some intention behind it, and thus that the changes have some meaningful relation to each other. In fact, if this assumption is completely misguided, recommendations based on change histories would degenerate to random prediction, which is clearly not the case.[2] Thus, given a subset Q of transaction C, a good recommendation algorithm should identify the complement $E = C/Q$ as the set of impacted items. Here E captures the ground truth on whether a change recommendation is truly relevant because it includes those artifacts that actually changed alongside Q. For this reason, E is used when evaluating the change recommendation generated for query Q.

Sampling strategy

Construction of the change scenarios involves two steps:

- sampling transactions
- generating queries from each sampled transaction.

[1] We currently extract method-level change information from files of C, C++, C#, and Java code.

[2] The chance of randomly predicting a correct method is $1/(\text{number of methods})$, which for any sizable system approaches zero.

Table E.1: Characteristics of the evaluated software systems (based on our extraction of the last 50 000 transactions for each of he systems).

Software System	History (in yrs)	Unique # files	Unique # artifacts	Avg. # artifacts in commit
CPython	12.05	7725	30090	4.52
Mozilla Gecko	1.08	86650	231850	12.28
Git	11.02	3753	17716	3.13
Apache Hadoop	6.91	24607	272902	47.79
HTTPD	19.78	10019	29216	6.99
Liferay Portal	0.87	144792	767955	29.9
Linux Kernel	0.77	26412	161022	5.5
MySQL	10.68	42589	136925	10.66
PHP	10.82	21295	53510	6.74
Ruby on Rails	11.42	10631	10631	2.56
RavenDB	8.59	29245	47403	8.27
Subversion	14.03	6559	46136	6.36
WebKit	3.33	281898	397850	18.12
Wine	6.6	8234	126177	6.68
Cisco Norway	2.43	64974	251321	13.62
Kongsberg Maritime	15.97	35111	35111	5.08

Software System	Languages used*
CPython	Python (53%), C (36%), 16 other (11%)
Mozilla Gecko	C++ (37%), C (17%), JavaScript (21%), 34 other (25%)
Git	C (45%), shell script (35%), Perl (9%), 14 other (11%)
Apache Hadoop	Java (65%), XML (31%), 10 other (4%)
HTTPD	XML (56%), C (32%), Forth (8%), 19 other (4%)
Liferay Portal	Java (71%), XML (23%), 12 other (4%)
Linux Kernel	C (94%), 16 other (6%)
MySQL	C++ (57%), C (18%), JavaScript (16%), 24 other (9%)
PHP	C (59%), PHP (13%), XML (8%), 24 other (20%)
Ruby on Rails	Ruby (98%), 6 other (2%)
RavenDB	C# (52%), JavaScript (27%), XML (16%), 12 other (5%)
Subversion	C (61%), Python (19%), C++ (7%), 15 other (13%)
WebKit	HTML (29%), JavaScript (30%), C++ (26%), 23 other (15%)
Wine	C (97%), 16 other (3%)
Cisco Norway	C++, C, C#, Python, Java, XML, other build/config
Kongsberg Maritime	C++, C, XML, other build/config

* languages used by open source systems are from `http://www.openhub.net`, percentages for the industrial systems are not disclosed.

We start by fetching the 50 000 most recent transactions from each subject system. From these, we then determine the 10 000 most recent transactions whose size is between 2 and 300. The minimum number, 2, ensures that there is always a potential impact set for any given query, while the maximum number, 300, covers at least 99% of all transactions while aiming to omit large changes such as licencing changes. From each sampled transaction C, the impact set E is randomly determined, but ensured to consist of at most ten items.

Ranking Rules

The rules mined by each algorithm all share the property that their support is at least one because we operate with an absolute minimal support constraint. By including "all rules" like this, we ensure that both high frequency as well as low frequency (rare) rules are included in the recommendations [39].

The support measure is not otherwise used for ranking. When support is used for ranking, special care needs to be taken as rare (infrequent) rules always rank lower then more frequent rules. This is a result of the *downward closure* property: any subset of a rule must have equal or larger support relative to the origin rule [40]. For example, given the two rules $r_1 = \{a\} \rightarrow \{x\}$ and $r_2 = \{a, b\} \rightarrow \{x\}$, r_2 cannot have higher support than r_1.

By using the confidence measure to rank rules, both rare and non-rare rules may be ranked highly. Still, the frequency of a pattern continues to inform the relevancy of a rule. To this end, recall that *the top 10 mean support* is included as a feature in our prediction model.

6.2 Evaluation of Relevance Prediction

Blocked cross validation

Last block validation is a frequently used scheme for evaluating prediction models. Here the data is split into two blocks, with the first block being used to train the model and the second block being used to evaluate it. This setup has the advantage of respecting the temporal nature of the data. However, a drawback is that not all data is used for both training and prediction [41]. To address this a traditional cross-validation setup may be used. However, doing so violates the temporal nature of time series data, and, in the worst case, may invalidate the results [41]. Because in time series data future data naturally depends on prior data, we use *blocked cross validation* to preserve the temporal order between training and evaluation. In our configuration we partition each set of transactions into ten equally sized blocks, preserving temporal order between blocks. We then train nine random forest classification models for each software-system and mining algorithm combination. As illustrated in

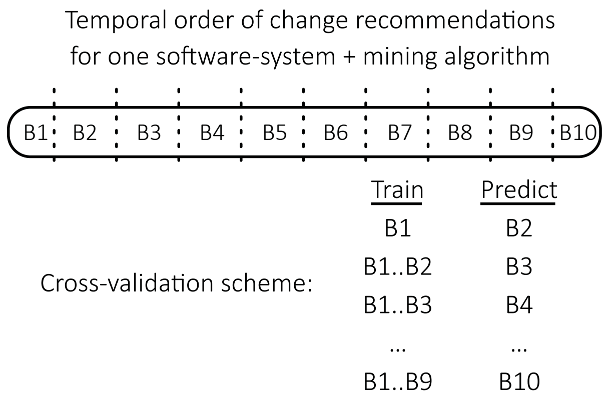


Fig. E.1: The blocked cross-validation scheme used in our study. Notice that all blocks except B1 and B10 are used for both training and prediction

Figure E.1, each random forest is trained on an incrementally larger subset of the available recommendations. In total, $16_{systems} * 2_{algorithms} * 9_{forests} = 288$ random forests are trained.

Measuring Relevance

The *random forest* and *confidence based* classification models have probabilistic interpretations. The confidence interestingness measure itself is given by the conditional likelihood $P(B|A)$, where B is the recommended artifact and A is one or more artifacts that changed (i.e., artifacts from the query) [12]. The maximum and mean confidence models use this information to capture the likelihood that a developer will act upon a given change recommendation. Here a "0" indicates very unlikely while a "1" indicates very likely. In the case of random forests, the likelihood is the result of the votes obtained from the collection of trees making up the random forest [8]. Each decision tree casts a single vote. As each tree is built from a random selection of change recommendations, the end likelihood is an indication of internal consistency within the data set for a particular scenario. In other words, if a certain scenario always results in a certain outcome, it is very likely that similar new scenarios will have the same outcome. Finally, in the *evaluation sets* used throughout our study we encode the possible classes in a similar way, the binary options are either 0 (not correct in top 10) or 1 (correct in top 10).

7 Results and Discussion

We organize the discussion of our results around the two research questions proposed in section 5. Throughout this section we will consider prediction performance for each individual software system, and for each of the mining

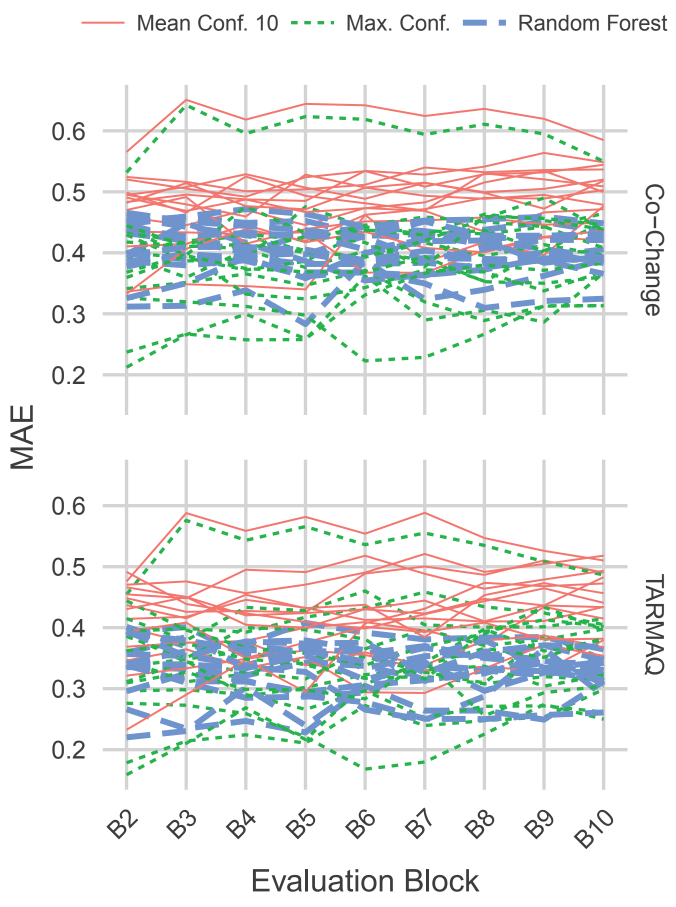


Fig. E.2: Descriptive view of errors using the MAE. Each line is a distinct software-system.

algorithms: Co-Change and Tarmaq. By doing so we get an indication of how generalizable the results are to other systems and other algorithms. In the following we briefly introduce each performance metric before presenting the corresponding results.

7.1 RQ 1: Comparison to confidence as a relevance predictor

While comparing the three classification models, we focus on two aspects of their relevance predictions:

1. the *error* with respect to the actual relevance, and
2. the *performance* across different classification thresholds.

We start with a descriptive view of the errors exhibited by each classification model, for this we use the Mean Absolute Error (MAE). In our case, MAE measures the mean absolute difference between the *actual* and *predicted* value for whether there is a correct artifact in the top ten artifacts of a recommendation. For example, given a certain feature set as input, the random forest

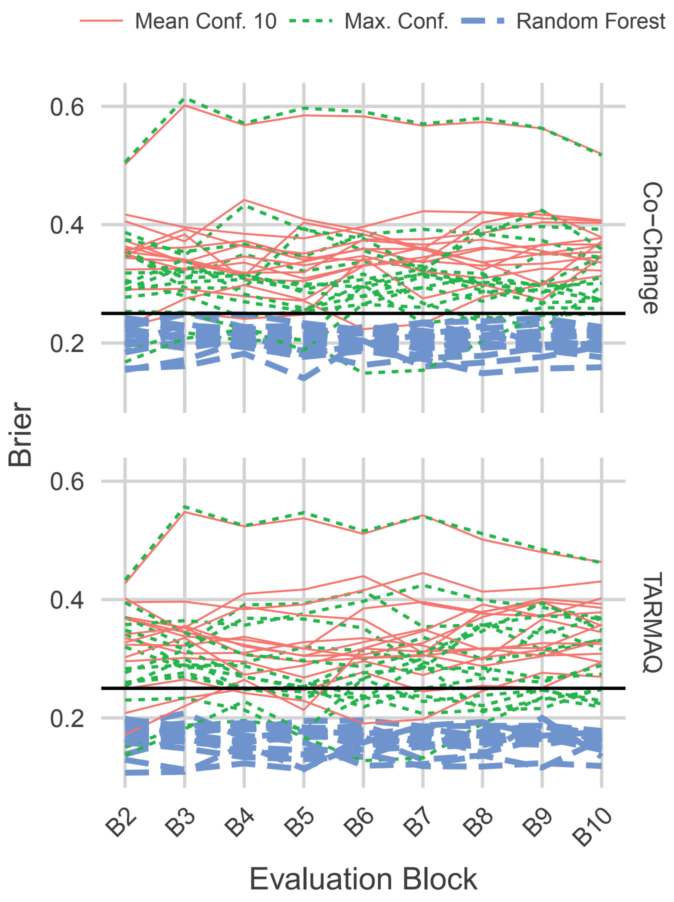


Fig. E.3: Accuracy of classification models using Brier scores. Each line is a distinct software-system. Models below the horizontal black line tend to classify correctly with regards to a 0.5 classification threshold.

might give the output *0.67*, indicating a 67% likelihood that the resulting recommendation will be relevant. If in actuality, we know that for this scenario there is indeed a correct artifact in the top ten the *prediction error* would be $1 - 0.67 = 0.33$ as positives are encoded as "1". Note that lower is better. In Figure E.2 we have provided the MAE across software systems for the two mining algorithms. For this first look at the data we have also preserved results for each *evaluation block*. This enables a view into error fluctuations across time. First, there is no apparent overall trend across the evaluation blocks. This is good news as it provides evidence that the analysis is stable across time. This is also supported by fitting linear regression lines (left out to minimize clutter). The *random forest* model shows less variance in error across systems and algorithms, while for some systems the *maximum confidence* model exhibits less overall error. For the change recommendations where the actual relevance was 1 (Correct in Top 10), the *maximum confidence* model frequently matches the prediction exactly and therefore minimizes the error for these recommendations.

In terms of *accuracy* of each classification model a *proper scoring rule* must be used [42]. For proper scoring rules, the maximum score is achieved if the *prediction distribution* exactly matches the *actual distribution*. One such scoring rule is the *brier score*. In the case of binary classification, which is our task, the brier score is the *mean squared error*:

$$BS = \frac{1}{N}\sum_{i=1}^{N}(p_i - a_i)^2$$

Where p_i is the predicted relevance for scenario i, and a_i is the actual relevance (1 or 0). Figure E.3 presents the Brier scores for each classification model across each evaluation block, software system, and mining algorithm. Note that lower is better. In the figure, the horizontal black line at y = 0.25 indicates the brier score of a *neutral classification model*. A neutral model always makes the prediction 0.5, where relevance and non-relevance are equally likely. Brier scores below the line indicate a prediction model that tends to be on the "right side of the midpoint". We clearly see the random forest model being consistently more accurate across algorithms and software systems.

While the *error* informs us about the overall fit of a prediction model, it does not capture performance across different *classification thresholds*. When classification models are used in practice, thresholds must be set to balance true positives against false positives and false negatives. To investigate the ability of our three prediction models in these terms, we consider the ROC curve and the Precision/Recall curve. First, the ROC curve in Figure E.4 plots the True Positive Rate ($TPR = \frac{TP}{TP+FN}$) against the False Positive Rate ($FPR = \frac{FP}{FP+TN}$) at classification thresholds from 0 to 1. It is immediately apparent that the confidence based models *do not meaningfully respond to different classification thresholds*, as data points are not evenly spread across the x-axis. Furthermore, there are strong linear relationships between their TPRs and FPRs. This was also reflected in corresponding Pearson correlation coefficients, where all coefficients were calculated to be *0.97* or higher. Intuitively, the effect we see is that as the classification threshold is lowered, the confidence models for recommendation relevance classify comparably increased amounts of recommendations both correctly and incorrectly. For example, both TPR and FPR increase similarly. Furthermore, observe that the range and domain of the TPR and FPR for the confidence models do not fully extend between 0 and 1. This is the result of a high percentage of change scenarios being given the relevance "1", and these scenarios being evenly split between True Positives (TPs) and False Positives (FPs). In other words, the lack of even distribution of data-points results in less variation in TPR and FPR, which again is reflected in the range and domain. To further support our findings we compared the partial Area Under the Curve (pAUC) between the ROC of the random forests and each of the confidence based

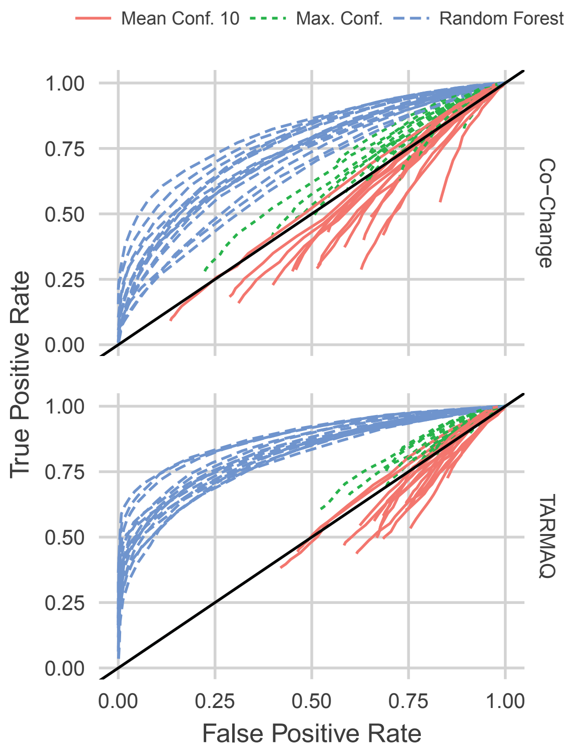


Fig. E.4: ROC curves for the prediction models trained for each software system and algorithm.

models. We used `roc.test` from the R package `pROC`, the significance test is based on bootstrapped sampling of multiple AUCs and computing their difference. Across all software systems and for both Co-Change and Tarmaq the pAUC were significantly larger ($p < 0.05$) for the random forest model. Thus, the random forest classifier consistently provide better estimates of relevance across various thresholds compared to the purely confidence based methods explored.

As laid out earlier, relevance prediction can be performed in a *background* thread that only notifies the developer when there is a high likelihood for a true positive recommendation. In this application, the positive class (correct in top ten) is therefore of the most interest. Notifications that there are *no relevant artifacts* would be of less use. With this view, the *precision* ($\frac{TP}{TP+FP}$) of the classification models become imperative. The task is to find an appropriate classification threshold that makes true positives likely (high precision), while still maintaining practicality in that recommendations can be regularly made (high recall). Figure E.5 shows the precision/recall curves for our three prediction models for each software system and mining algorithm. First, the abnormality in slope, range and domain for the confidence models can again be attributed to the weak connection to threshold-changes. Furthermore, while one usually expects a decrease in precision as recall increases, this does not necessarily *need* to be the case. The trends for the confidence

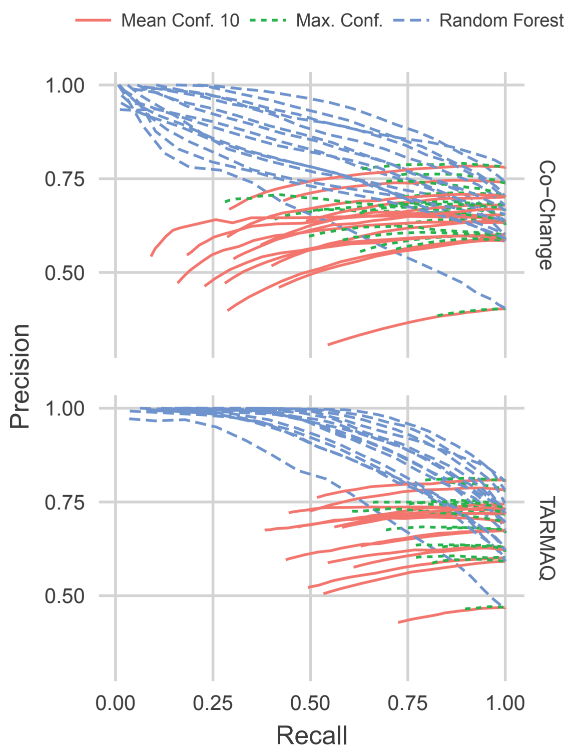


Fig. E.5: Precision/Recall curves for the prediction models trained for each software system and algorithm.

models in Figure E.5 are the result of having slightly higher concentrations of positive classifications than negative classifications on lower thresholds. As thresholds are lowered further, more recommendations become TPs, and the ratio between TPs and FPs actually increases. An implication of this is that at least for the confidence measure, its value cannot be used directly as an indication of relevancy.

Turning to the random forest models, these exhibit greater defined behavior, where negative classifications are primarily located in lower likelihood thresholds, thus precision decreases as recall increases. In terms of recommending concrete classification thresholds for our random forest model we suggest that this should be adjusted with respect to the *system domain knowledge* of the developer for which the recommendation is made. In Table E.2 we have provided the mean precision and recall across software systems for the example thresholds at 0.5 and 0.9. As developers become more acquainted with a system, they should also be able to better differentiate relevant and non-relevant recommendations. As such, experienced developers might afford a higher rate of recall at the cost of lower precision. A classification threshold of 0.50 becomes reasonable for this group, assuming TARMAQ is used. For inexperienced developers one wants to minimize confusion, and therefore maximize the precision of change recommendations. Thus, a threshold such as 0.9 might be appropriate for these developers, resulting in

Table E.2: Examples of Precision and Recall for the random forest classification model. The Standard Deviation (SD) captures fluctuations between software systems.

		Classification Threshold			
		0.5		0.9	
Algorithm		Mean	SD	Mean	SD
Co-Change	Precision	0.7749	± 0.0754	0.9710	± 0.0204
	Recall	0.6835	± 0.0458	0.1003	± 0.0735
Tarmaq	Precision	0.8677	± 0.0619	0.9929	± 0.0082
	Recall	0.7348	± 0.0324	0.2771	± 0.1016

change recommendations only having false positives in about 1 to 3 per 100 recommendations.

7.2 RQ 2: Analysis of features

Having empirically established that the random forest classifications models are superior at predicting change recommendation relevance, we next consider which features bring the most value to the models. Breiman introduced the concept of *variable importance* for random forests [8]. Once the decision trees of the random forests have been built, the process can self-assess the importance of each feature. The basic idea is to observe if randomly permuting a feature changes prediction performance [8]. Averaging the accuracy changes over all trees gives the *mean decrease in accuracy* (when permuted) for each feature.

The corresponding plot for the features included in our model is provided in Figure E.6. For two of the studied software systems (KM and Rails) we have only file-level change information. These two systems are shown using dotted lines. Naturally, permuting the "*Number of methods changed*" feature does not change accuracy for these two systems, as the value is always 0, as reflected in Figure E.6.

To begin with Tarmaq was constructed to produce a focused recommendation that matches the query as closely as possible [36]. This is evident in the figure where the "*Number of rules generated*" being an essential feature for Tarmaq. Thus for Tarmaq, variation in the number of rules generated meaningfully correlates with recommendation relevance; if a large subset of the query has changed with something else in the past, this results in fewer possible rule antecedents and thus fewer rules, which evidently increases the likelihood of a relevant recommendation. For Co-Change this feature has less importance. For the remaining features, Figure E.6 shows a rather clear picture; the query based features (the bottom four) are the least important, the attributes they represent are better captured by other features. Of the interestingness measure based features the *"Top 10 mean confidence"* proved

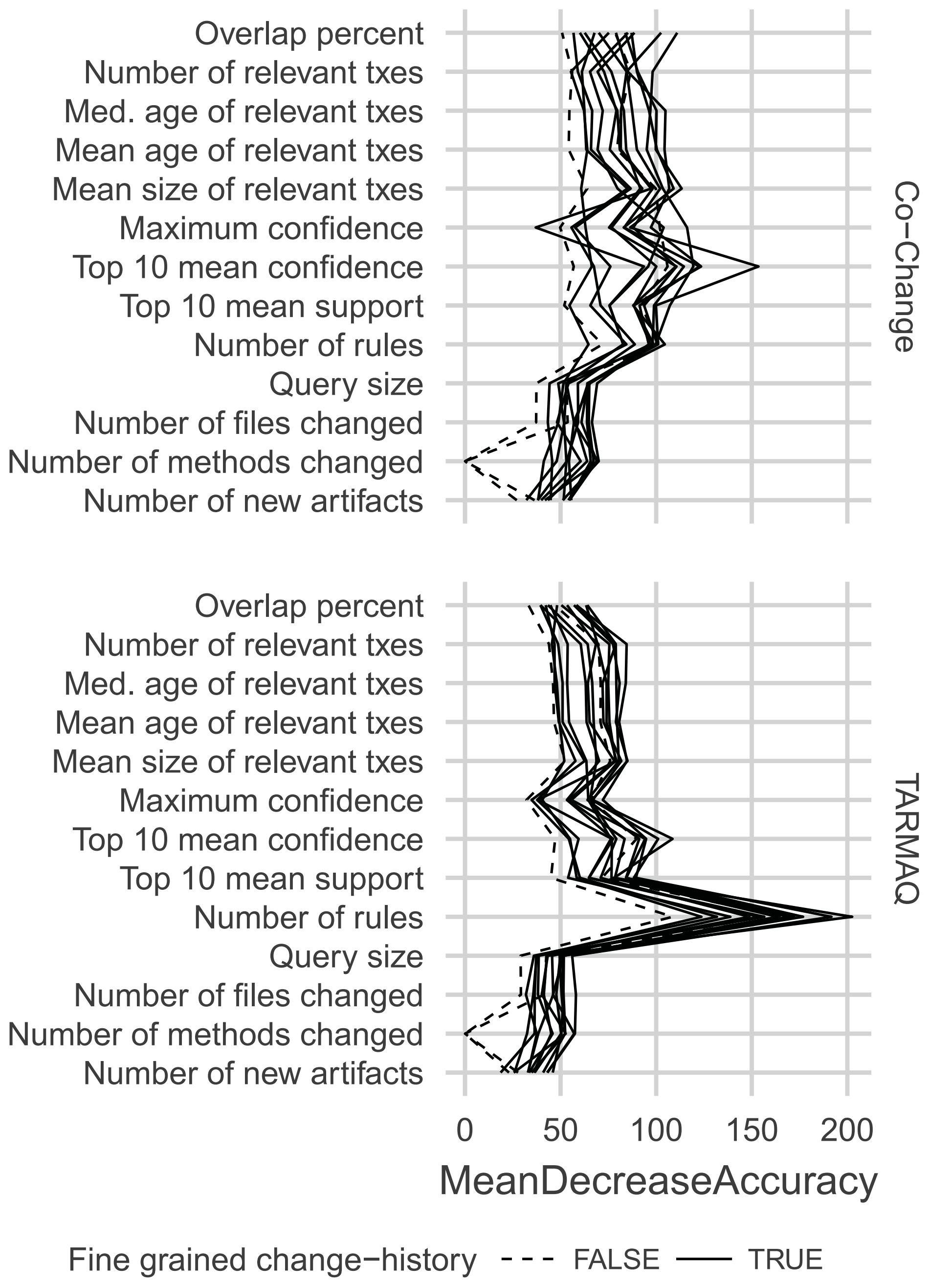


Fig. E.6: Feature importance as determined by mean decrease in accuracy. Each line is a separate software system.

most useful. Finally, all history related features are comparably important.

The high degree of co-variance between software systems suggests that *model transfer* to other systems is viable. That is, classification models learned on one or more systems, should be reusable for a new (unknown) system. If this can be shown to work reliably, systems that are early in their evolution can still benefit from models generated from more mature systems. However, care must be taken to adapt the feature variation of the random forest to fit the variation found in new software system.

7.3 Threats to Validity

Implementation: We implemented and thoroughly tested all algorithms and our technique for model classification in Ruby and R respectively, However, we can not guarantee the absence of implementation errors.
Variation in software systems: We have sought to obtain generalizable results by evaluating over a range of heterogeneous software systems, however, we also uncovered that relevance prediction performance varies between systems. Future work should investigate the effects of system characteristics on prediction performance.
Mining Algorithms studied: In our evaluation we have studied classification models for recommendations generated using both the Co-Change and Tarmaq mining algorithms. For both algorithms we achieve strong results. However, we acknowledge that comparable results cannot be guaranteed for other mining algorithms.
Using other interestingness measures: In our study we focused on the *confidence* interestingness measure, thus our results are limited to this measure. As such, future work should investigate the use of other interestingness measures, both for comparison to the random forest predictor, as well as being included as part of the model. We also envision that a variation of the relevance prediction presented here might be an interestingness measure recommender, thus essentially creating an ensemble of measures where the most relevant is used at a given time.
Recommendations used for training and evaluation: We train and evaluate our classification model over a constructed set of change recommendations. Each recommendation is the result of executing randomly sampled a query from an existing transaction where the complement of the query and the source transaction is used as the *expected outcome*. However, this approach does not account for the actual order in which changes were made before they were committed to the versioning system. As a result, it is possible that queries contain elements that were actually changed later in time than elements of the expected outcome. As such, we cannot guarantee that recommendations used in training and evaluation exactly reflect each system's evolution.

8 Concluding Remarks

Change recommendation is clearly an asset to a software developer maintaining a complex system. However, its practical adoption faces two challenges: recommendations must be both accurate and relevant. We believe that both challenges can be effectively addressed using *historically proven change recommendations*.

This paper shows that random forest classification using the 12 features that describe aspects of the *change set* (query), *change history* (transactions) and generated *change recommendations* is viable. We compare the random forest model against the state-of-the-art (based on *confidence*). We evaluate our approach in a large empirical study across 16 software systems and two change recommendation algorithms. Our findings are as follows:
Finding 1: The random forest classification model consistently outperforms the confidence based models in terms of accuracy (Brier scores).
Finding 2: The random forest classification model achieves significantly larger area under ROC curve than both confidence based models.
Finding 3: While the confidence measure is appropriate for ranking of artifacts, the values themselves should not be interpreted in isolation as overall estimates of recommendation relevance.
Finding 4: The importance of model features may varies between algorithms. For example, the relevance of Tarmaq recommendations is best predicted by considering the number of rules generated, while this feature is less important for Co-Change. However, the remaining features studies showed consistent importance between algorithms.

Directions for Future Work

Looking forward, we hope to study the effects of using stricter and looser definitions of relevance (e.g., relevant if correct in top three vs top twenty). Furthermore, rather than classification, relevance can also be studied as a regression problem, predicting other recommendation metrics such as precision, recall, average precision etc. In addition we plan to study the behavior of relevance classification models over other interestingness measures, and investigate the viability of model transfer between software systems. Finally, we plan to look to improve the classification model by including features such as the number of relevant transactions authored by core contributors vs occasional contributors, the weighting recent relevant transactions higher than older transactions, and the text similarity scores between change set and relevant transactions.
Acknowledgement: This work is supported by the Research Council of

Norway through the EvolveIT project[3] and the Certus SFI[4]. Dr. Binkley is supported by NSF grant IIA-1360707 and a J. William Fulbright award.

References

[1] S. Bohner and R. Arnold, *Software Change Impact Analysis*. CA, USA: IEEE, 1996.

[2] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 1998, pp. 190–198. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=738508

[3] D. Beyer and A. Noack, "Clustering Software Artifacts Based on Frequent Common Changes," in *International Workshop on Program Comprehension (IWPC)*. IEEE, 2005, pp. 259–268. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1421041

[4] R. Robbes, D. Pollet, and M. Lanza, "Logical Coupling Based on Fine-Grained Change Information," in *Working Conference on Reverse Engineering (WCRE)*. IEEE, 2008, pp. 42–46. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4656392

[5] H. Gall, M. Jazayeri, and J. Krajewski, "CVS release history data for detecting logical couplings," in *International Workshop on Principles of Software Evolution (IWPSE)*. IEEE, 2003, pp. 13–23. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1231205

[6] M. B. Zanjani, G. Swartzendruber, and H. Kagdi, "Impact analysis of change requests on source code based on interaction and commit histories," in *International Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 162–171. [Online]. Available: http://dl.acm.org/citation.cfm?id=2597096http://dx.doi.org/10.1145/2597073.2597096

[7] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?" in *Proceedings of the 2011 International Symposium on Software Testing and Analysis - ISSTA '11*. New York, New York, USA: ACM Press, 2011, p. 199. [Online]. Available: http://portal.acm.org/citation.cfm?doid=2001420.2001445

[3] #221751/F20

[4] #203461/030

[8] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[9] P.-N. Tan, V. Kumar, and J. Srivastava, "Selecting the right interestingness measure for association patterns," in *International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2002, p. 32. [Online]. Available: http://portal.acm.org/citation.cfm?doid=775047.775053

[10] K. McGarry, "A survey of interestingness measures for knowledge discovery," *The Knowledge Engineering Review*, vol. 20, no. 01, p. 39, 2005.

[11] L. Geng and H. J. Hamilton, "Interestingness measures for data mining," *ACM Computing Surveys*, vol. 38, no. 3, sep 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1132960.1132963

[12] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD International Conference on Management of Data*. ACM, 1993, pp. 207–216. [Online]. Available: http://portal.acm.org/citation.cfm?doid=170035.170072

[13] L. Moonen, S. Di Alesio, D. W. Binkley, and T. Rolfsnes, "Practical Guidelines for Change Recommendation using Association Rule Mining," in *International Conference on Automated Software Engineering (ASE)*. Singapore: IEEE, sep 2016.

[14] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," *Proceedings of the 23th International Conference on Machine Learning*, pp. 161–168, 2006. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143865

[15] P. Lenca, P. Meyer, B. Vaillant, and S. Lallich, "On selecting interestingness measures for association rules: User oriented description and multiple criteria decision aid," *European Journal of Operational Research*, vol. 184, no. 2, pp. 610–626, jan 2008. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0377221706011465

[16] T.-d. B. Le and D. Lo, "Beyond support and confidence: Exploring interestingness measures for rule-based specification mining," in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 331–340. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7081843

[17] W. Cheetham, "Case-Based Reasoning with Confidence," in *European Workshop on Advances in Case-Based Reasoning (EWCBR)*, ser. Lecture Notes in Computer Science, vol 1898. Springer, 2000, pp.

15–25. [Online]. Available: http://www.springerlink.com/content/5dt7vmf5je4ngyq7http://link.springer.com/10.1007/3-540-44527-7{_}3

[18] W. Cheetham and J. Price, "Measures of Solution Accuracy in Case-Based Reasoning Systems," in *European Conference on Case-Based Reasoning (ECCBR)*, ser. Lecture Notes in Computer Science, vol 3155. Springer, 2004, pp. 106–118. [Online]. Available: http://www.springerlink.com/content/mkl87lfx92jr02k0http://link.springer.com/10.1007/978-3-540-28631-8{_}9

[19] T.-D. B. Le, F. Thung, and D. Lo, "Predicting Effectiveness of IR-Based Bug Localization Techniques," in *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, nov 2014, pp. 335–345. [Online]. Available: http://ieeexplore.ieee.org/document/6982639/

[20] T.-D. B. Le, D. Lo, and F. Thung, "Should I follow this fault localization tool's output?" *Empirical Software Engineering*, vol. 20, no. 5, pp. 1237–1274, oct 2015. [Online]. Available: http://link.springer.com/10.1007/s10664-014-9349-1

[21] T. Rolfsnes, L. Moonen, S. Di Alesio, R. Behjati, and D. W. Binkley, "Improving change recommendation using aggregated association rules," in *International Conference on Mining Software Repositories (MSR)*. ACM, 2016, pp. 73–84. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2901739.2901756

[22] H. Toivonen, M. Klemettinen, P. Ronkainen, K. Hätönen, and H. Mannila, "Pruning and Grouping Discovered Association Rules," in *Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases*, Heraklion, Crete, Greece, 1995, pp. 47–52.

[23] B. Liu, W. Hsu, and Y. Ma, "Pruning and summarizing the discovered associations," in *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 1999, pp. 125–134. [Online]. Available: http://portal.acm.org/citation.cfm?doid=312129.312216

[24] S. Kannan and R. Bhaskaran, "Association Rule Pruning based on Interestingness Measures with Clustering," *Journal of Computer Science*, vol. 6, no. 1, pp. 35–43, dec 2009.

[25] M. J. Zaki, "Generating non-redundant association rules," in *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2000, pp. 34–43. [Online]. Available: http://portal.acm.org/citation.cfm?doid=347090.347101

[26] E. Baralis, L. Cagliero, T. Cerquitelli, and P. Garza, “Generalized association rule mining with constraints,” *Information Sciences*, vol. 194, pp. 68–84, 2012. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0020025511002659

[27] O. Maimon and L. Rokach, *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Springer, 2010. [Online]. Available: http://books.google.com/books?hl=en{&}lr={&}id=S-XvEQWABeUC{&}oi=fnd{&}pg=PR21{&}dq=Data+Mining+and+knowledge+discovery+handbook{&}ots=LBVkfoBx6S{&}sig=u6c1n2kopRhLrbpg5M0FhvYhFqk{%}5Cnhttp://www.springerlink.com/index/10.1007/978-0-387-09823-4http://link.springer.com/1

[28] Z. Zheng, R. Kohavi, and L. Mason, “Real world performance of association rule algorithms,” in *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2001, pp. 401–406. [Online]. Available: http://portal.acm.org/citation.cfm?doid=502512.502572

[29] W. Lin, S. A. Alvarez, and C. Ruiz, “Efficient Adaptive-Support Association Rule Mining for Recommender Systems,” *Data Mining and Knowledge Discovery*, vol. 6, no. 1, pp. 83–105, 2002. [Online]. Available: http://link.springer.com/10.1023/A:1013284820704

[30] N. Jiang and L. Gruenwald, “Research issues in data stream association rule mining,” *ACM SIGMOD Record*, vol. 35, no. 1, pp. 14–19, mar 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1121995.1121998

[31] L. Moonen, S. Di Alesio, T. Rolfsnes, and D. W. Binkley, “Exploring the Effects of History Length and Age on Mining Software Change Impact,” in *IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, sep 2016, pp. 207–216.

[32] P. Resnick and H. R. Varian, “Recommender systems,” *Communications of the ACM*, vol. 40, no. 3, pp. 56–58, mar 1997. [Online]. Available: http://portal.acm.org/citation.cfm?id=245108.245121http://portal.acm.org/citation.cfm?doid=245108.245121

[33] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, “Mining version histories to guide software changes,” *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1463228

[34] R. Srikant, Q. Vu, and R. Agrawal, “Mining Association Rules with Item Constraints,” in *International Conference on Knowledge Discovery and Data Mining (KDD)*. AASI, 1997, pp. 67–73.

[35] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. L. Collard, “Blending conceptual and evolutionary couplings to support change impact analysis in source code,” in *Working Conference on Reverse Engineering (WCRE)*, 2010, pp. 119–128.

[36] T. Rolfsnes, S. Di Alesio, R. Behjati, L. Moonen, and D. W. Binkley, “Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis,” in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, mar 2016, pp. 201–212. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7476643

[37] S. Eick, T. L. Graves, A. Karr, J. Marron, and A. Mockus, “Does code decay? Assessing the evidence from change management data,” *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 1–12, 2001. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=895984

[38] A. T. T. Ying, G. Murphy, R. T. Ng, and M. Chu-Carroll, “Predicting source code changes by mining change history,” *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1324645

[39] M.-C. Tseng and W.-Y. Lin, “Mining Generalized Association Rules with Multiple Minimum Supports,” in *Lecture Notes in Computer Science (LNCS)*, 2001, vol. 2114, pp. 11–20. [Online]. Available: http://link.springer.com/10.1007/3-540-44801-2{_}2

[40] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” in *International Conference on Very Large Data Bases (VLDB)*, 1994, pp. 487–499.

[41] C. Bergmeir and J. M. Benítez, “On the use of cross-validation for time series predictor evaluation,” *Information Sciences*, vol. 191, pp. 192–213, may 2012. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2011.12.028http://linkinghub.elsevier.com/retrieve/pii/S0020025511006773

[42] A. Buja, W. Stuetzle, and Y. Shen, “Loss functions for binary class probability estimation and classification: structure and application,” 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.184.5203{&}rep=rep1{&}type=pdf